Proyecto Fin de Carrera:
Sistema para proteger la privacidad de los usuarios de los motores de búsqueda de Internet

José Manuel Puga Moreira
Ingeniería en Informática

Jordi Castellà-Roca
Consultor

Fecha de entrega
12 de Junio del 2011

Gracias a mis padres, a mi hermana y en general a mi familia gallega y catalana, por confiar en mi y en que algún día llegaría este momento.

Gracias a Jordi, mi tutor, por solucionar rápidamente todas las dudas que fueron surgiendo en este tiempo.

Mención especial a Carina por su apoyo, su paciencia, sus ánimos, y por hacer más llevaderos estos años de intenso trabajo. Sin ella no lo hubiese conseguido. Gracias.

Este proyecto se lo dedico a Manuel y Hermencina, mis padres.

Resumen

El incremento de usuarios de Internet en la última década ha sido cuanto menos espectacular. La Web 2.0 ha supuesto una revolución en las comunicaciones, en la forma de crear y compartir información, en la colaboración entre usuarios, etc.. Si se tiene en cuenta que esto supone un mayor número de contenidos en la red, es necesario que los buscadores de Internet (WSE) compitan por ofrecer un mejor servicio a sus usuarios. Y esto pasa por mostrar los mejores resultados en las primeras páginas.

Esta mejora del servicio deriva en la necesidad de los WSE de “conocer” en cierta manera a quién está realizando la búsqueda. Para ello, elaboran un perfil del usuario con sus búsquedas y de esta forma, según la consulta que envíe el usuario puedan ofrecer una mejor respuesta en base a su historial.

Pero este perfil supone una amenaza a la privacidad del usuario. Los WSE mantienen información de sus clientes que puede contener datos tan personales como el nombre del usuario, sus apellidos, su DNI, aficiones, etc..

Existen propuestas actualmente que impiden que los WSE elaboren perfiles de los usuarios, manteniendo la privacidad de los mismos, sin embargo estas herramientas se caracterizan por sus elevados tiempos de respuesta en las búsquedas. Además el hecho de mantener el anonimato de los usuarios impide que los WSE puedan ofrecer un servicio más personalizado y por lo tanto ofrecer las posibles mejores respuestas en las primeras páginas.

En este proyecto se ha implementado un método alternativo que hace uso de las redes sociales para mantener la privacidad de los usuarios pero a la vez permitir al WSE elaborar un perfil útil para poder ofrecer este mejor servicio al usuario.

Índice de contenidos

1 INTRODUCCIÓN 11

1.1 JUSTIFICACIÓN DEL PFC Y CONTEXTO EN EL CUAL SE DESARROLLA 11

1.2 OBJETIVOS DEL PROYECTO 13

1.3 ENFOQUE Y MÉTODO SEGUIDO 15

1.4 PLANIFICACIÓN 15

1.5 PRODUCTOS OBTENIDOS 16

1.6 ESTRUCTURA DE LA MEMORIA 16

2 DESCRIPCIÓN DEL ESQUEMA 18

2.1 ARQUITECTURA 18

2.2 TIPOS DE USUARIOS 19

2.3 DESCRIPCIÓN DEL PROTOCOLO 20

2.3.1 PROBABILIDAD DE ENVÍO P_S 21

2.3.2 FUNCIÓN DE SELECCIÓN DE USUARIO Ψ 21

2.3.3 FUNCIÓN DE REENVÍO DE CONSULTA Ψ_f 23

2.3.4 FUNCIÓN DEL NIVEL DE EGOÍSMO $\acute{Y}$ 24

2.4 PRIVACIDAD DE LOS USUARIOS 26

2.4.1 FIRMA DIGITAL 26

2.4.2 INTEGRACIÓN EN EL SISTEMA 27

3 DISEÑO E IMPLEMENTACIÓN 29

3.1 DECISIONES DE DISEÑO 29

3.1.1 LENGUAJE DE PROGRAMACIÓN 29

3.1.2 COMUNICACIONES 31

3.2 DISEÑO DE LA ARQUITECTURA. COMPONENTES 32

3.3 DIAGRAMA DE CLASES 34

3.3.1 SERVER 35

3.3.2 SERVERTHREAD 36

3.3.3 CLIENT 37

3.3.4 PROTOCOL 38

3.3.5 QUERY 39

3.3.6 USER 40

3.3.7 WSE 41

3.3.8 RESPONSE 41

3.3.9 XML 41

3.3.10 CRYPTO 41

4 PRUEBAS 42

4.1 RED SOCIAL UTILIZADA 42

4.2 METODOLOGÍA 44

4.3 CARGA DE DATOS EN LA BASE DE DATOS 46

4.3.1 READGRAPH 47

4.3.2 READGRAPHGUI 48
4.3.3 DLFILTER 48
4.4 CLASES “STATISTICS” Y “TEST” 48

5 RESULTADOS DE LAS SIMULACIONES 49
5.1 PROBABILIDADES FINALES 49
5.1.1 COMPORTAMIENTO NORMAL 49
5.1.2 COMPORTAMIENTO EGOÍSTA 51
5.2 DISTRIBUCIÓN DE CONSULTAS 52
5.2.1 COMPORTAMIENTO NORMAL 52
5.2.2 COMPORTAMIENTO EGOÍSTA 56
5.3 NÚMERO DE SALTOS 60
5.3.1 RED DE 20 USUARIOS 60
5.3.2 RED DE 100 USUARIOS 61
5.4 TIEMPO DE PROCESO DEL USUARIO 61

6 CONCLUSIONES 64
6.1 PRIVACIDAD 64
6.2 RESULTADOS DE LA IMPLEMENTACIÓN 65
6.3 PROBLEMAS ENCONTRADOS 66
6.4 TRABAJO FUTURO 67

7 BIBLIOGRAFÍA 69

8 ANEXO 1. INSTALACIÓN DE LA APLICACIÓN 71
8.1 BASE DE DATOS 71
8.2 IMPORTACIÓN DE LA RED SOCIAL 76

9 ANEXO 2. EJECUCIÓN DE UN TEST 79
9.1 INICIO DEL SERVIDOR DE LA RED SOCIAL 79
9.2 INICIO DEL SCRIPT DE PRUEBAS 80
9.3 REALIZACIÓN DE UNA PRUEBA 81

Índice de figuras

Figura 1.1 – Planificación del proyecto 16
Figura 2.1 – Envío de una consulta q con certificados de transacción 28
Figura 3.1 – Esquema de la interacción entre componentes 34
Figura 3.2 – Diagrama de clases 35
Figura 3.3 – Esquema XML de una consulta firmada 40
Figura 3.4 – Grafo que representa las relaciones entre los usuarios de la red social Twitter 43
Figura 3.5 – Red de 20 usuarios utilizada para las pruebas del protocolo 43
Figura 3.6 – Número de amigos de los usuarios de la red social 44
Figura 5.1 – Variables que influyen en el cálculo del tiempo de una consulta en un entorno real...62
Figura 8.1 – Inicio de los servicios Apache y MySql 72
Figura 8.2 – Interfaz web de PHPMyAdmin 72
Figura 8.3 – Apartado de “Privilegios” de la base de datos “sn_server” 73
Figura 8.4 – Formulario para agregar un nuevo usuario a la base de datos 74
Figura 8.5 – Selección de los privilegios del nuevo usuario 74
Figura 8.6 – Contenido del fichero “script_BBDD.sql” 75
Figura 8.7– Confirmación del estado de la importación 76
Figura 8.8 – Interfaz de la aplicación ReadGraph 77
Figura 8.9 – Selección del fichero que contiene los datos a importar 77
Figura 8.10 – Finalización exitosa de la importación 78
Figura 9.1 – Inicio del servidor 80
Figura 9.2 – Menú de la aplicación de Test 81
Figura 9.3 – Configuración de las opción de ejecución del test 82

Índice de tablas

Tabla 2.1 – Posición media, varianza y disviación obtenidas con diferentes Θ intervalos 23
Tabla 2.2 – Porcentaje de usuarios expuestos para diferentes valores de ζ 25
Tabla 5.1 – Probabilidades finales de los usuarios de la red social 50
Tabla 5.2 – Probabilidades finales con un comportamiento egoísta 51
Tabla 5.3 – Número de consultas enviadas al WSE y creadas por un determinado usuario . . .53-55
Tabla 5.4 – Número de consultas enviadas al WSE por cada usuario y creadas por un
usuario en concreto 57-59

Capítulo 1

Introducción

En el presente capítulo se enumeran las justificaciones en las que se basa el proyecto (apartado 1.1) y también los objetivos que se desean conseguir con su implementación, que se pueden observar en la sección 1.2. A continuación se describe el enfoque del proyecto (apartado 1.3) y la elaboración del plan de trabajo (sección 1.4). El capítulo concluye redactando los productos obtenidos en el apartado 1.5 y describiendo brevemente el contenido de la memoria en la sección 1.6.

1.1 Justificación del PFC y contexto en el cual se desarrolla

Para acceder a la información disponible en Internet, los usuarios utilizan un explorador web que permite el acceso a dicha información introduciendo una URL determinada. Sin embargo, dada la inmensa cantidad de datos almacenados en millones de servidores de Internet, es necesaria la existencia de los buscadores de Internet, que permiten encontrar una información específica mediante la introducción de una serie de términos relacionadas con dicha información.

Algunos estudios [1] detallan que el 68% de los usuarios de los buscadores seleccionan un resultado de búsqueda en la primera página de los resultados y que además, el 92% lo hace dentro de las tres primeras páginas de resultados. Por lo tanto, para ofrecer una mejor experiencia de usuario, los motores de búsqueda web deben poner en la primera página de resultados aquellos enlaces que crean más interesantes para sus clientes.

Con este fin y el de ofrecerle un mejor servicio, los motores de búsqueda almacenan información de cada consulta realizada por un usuario para crear un

perfil del mismo. De esta forma una persona que previamente haya realizado una búsqueda de “Dioses romanos” y que realiza una búsqueda posterior de la palabra “Júpiter”, hará que el motor de búsqueda consulte el perfil almacenado de dicho individuo asumiendo que con “Júpiter” se refiere al Dios romano y no al planeta del Sistema Solar, ofreciendo dichos resultados en las primeras páginas.

El proceso de mejorar la calidad de las respuestas ofrecidas por los motores de búsqueda de Internet utilizando perfiles de usuarios recibe el nombre de “*búsqueda personalizada*”.

Sin embargo, para muchos usuarios esto supone una violación de su privacidad, ya que para la elaboración de su perfil el buscador puede almacenar datos tan personales como DNI, nombre (¿quién no ha buscado en Internet su nombre o DNI?), intereses, enfermedades, etc.

Para solucionar este problema se podría pensar que existe una forma sencilla de proveer anonimato a los usuarios de los buscadores de Internet, que sería la de utilizar una IP dinámica y un navegador sin *cookies*. Sin embargo, hay que tener claro que la renovación de la dirección IP no depende del usuario sino de la operadora de red, pudiendo asignar la misma IP a la misma dirección MAC (Media Access Control). Señalar también que algunos usuarios utilizan IP estáticas, y por último que un navegador sin *cookies* pierde su usabilidad en numerosas aplicaciones web, por lo que no sería asequible para determinados usuarios.

Es evidente que los perfiles son necesarios para que los motores de búsqueda puedan ofrecer un servicio eficiente a sus clientes, por lo que existe una relación entre el nivel de privacidad y la calidad del servicio ofrecido.

Actualmente existen herramientas que protegen la privacidad de los usuarios, aunque estas soluciones se caracterizan o bien por los altos costes en términos de computación y comunicación o bien porque la implementación realizada para asegurar su privacidad impide al buscador crear un perfil del usuario y como consecuencia, no poder ofrecerle respuestas más adecuadas a sus términos de

búsqueda. Como referencia, en el sistema propuesto en [15], la mejor respuesta conseguida es de 5,84 segundos. En [11] el protocolo UUP logra conseguir respuestas con un tiempo de 5,2 segundos. Y por último, utilizando el "*Tor Project*" en [16], las búsquedas como media son 25 veces más lentas que una búsqueda directa.

Por todo lo expuesto anteriormente, este proyecto se enmarca en el contexto de la seguridad informática, concretamente en el de mantener la privacidad de los usuarios que utilizan los motores de búsqueda de Internet. En este sentido, el sistema implementado no privará a los motores de búsqueda de crear un perfil del usuario, sino que lo que conseguirá es que el buscador cree un perfil distorsionado del mismo. Para llevarlo a cabo, la implementación tiene en cuenta los siguientes aspectos:

- La proliferación de los ordenadores personales con banda ancha conectados a Internet, lo que implica que puedan estar conectados permanentemente a la red.
- El auge de las redes sociales. El sistema se basará en el concepto de red social para distorsionar el perfil de un usuario.

De esta forma, este proyecto se basa en las siguientes justificaciones:

- La primera y principal es la de obtener un sistema que permita mantener la privacidad de un usuario sin que ello afecte en demasía a los resultados de las búsquedas que los buscadores conseguirían teniendo un perfil del usuario.
- La segunda es la posibilidad de conseguir un sistema que mejore los tiempos de latencia de las consultas con respecto a las herramientas existentes para la protección de la privacidad de los usuarios de los motores de búsqueda de Internet.

1.2 Objetivos

La finalidad principal del presente proyecto es la de implementar un sistema que utilice las redes sociales para proveer un perfil “distorsionado” para los motores de búsqueda, sin necesidad de ningún cambio en el lado del servidor ni de que éste colabore con los usuarios para mantener su privacidad. Esto se conseguirá utilizando el concepto de grupo provisto por las redes sociales.

Un esquema del funcionamiento del sistema, a grandes rasgos, es el siguiente: cuando un usuario crea una consulta, éste decidirá si enviarla directamente al motor de búsqueda o si la envía a uno de sus vecinos de la red social, y éste a su vez decidirá si enviarla al motor de búsqueda o si la reenvía a uno de sus vecinos. Así sucesivamente hasta que un vecino acepte enviar la consulta al motor de búsqueda. En definitiva, se trata de que las consultas generadas por un usuario sean distribuidas uniformemente por la red social y enviadas por algún vecino al buscador web. El esquema y funcionamiento completo del sistema se detallará en el siguiente capítulo.

Con este sistema no se evita que el motor de búsqueda cree un perfil de cada usuario, pero sí se consigue evitar que el buscador cree un perfil exacto, obteniendo en su lugar uno distorsionado, ya que contendrá consultas de los miembros del grupo de la red social. Asumiendo que un usuario comparte intereses con sus vecinos de la red social, también se puede aceptar que un perfil distorsionado con consultas de los vecinos es útil (a un cierto nivel) para que el motor de búsqueda pueda ofrecer un servicio adecuado.

La implementación del protocolo descrito tiene los siguientes objetivos:

- En primer lugar, debe ser utilizable en la práctica. Se tendrá en cuenta que los grupos de usuarios están ya creados, por lo que el sistema no tendrá que crear grupos para el envío de consultas. Este hecho representa una significante reducción de tiempo en comparación con otras propuestas actuales.

- En segundo lugar, el motor de búsqueda no será capaz de obtener un perfil exacto de los usuarios. Sin embargo, los perfiles deben ser útiles (a un determinado nivel) para que los motores de búsqueda puedan proveer un servicio adecuado al usuario.
- Finalmente, el hecho de que una consulta de un usuario pueda ser enviada al motor de búsqueda por otro vecino de la red social implica que el buscador no sabrá quién ha generado realmente la consulta. Sin embargo, si un usuario malintencionado genera consultas que vulneran la ley, y los responsables gubernamentales intervienen para conocer de donde procede dicha consulta, el sistema podrá demostrar inequívocamente qué usuario ha generado esa consulta. Esto se consigue utilizando la “firma electrónica avanzada”. Este tipo de firma digital tiene un fuerte valor judicial: garantiza autenticación, integridad, confidencialidad y no repudio.

1.3 Enfoque del PFC y método seguido

El proyecto se ha dividido en diferentes fases con el fin de realizar una implementación incremental, es decir, en cada fase se ha implementado la funcionalidad correspondiente, y tras las pruebas pertinentes en cada una de ellas se realizaba la integración con la fase anterior.

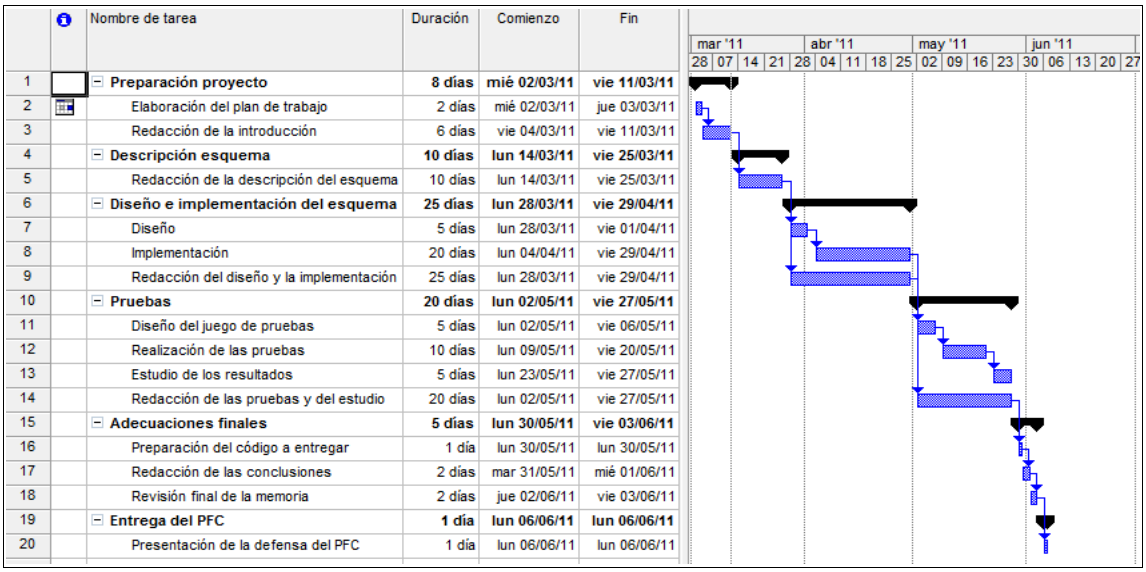
Las fases son las siguientes:

- Instalación del software (Eclipse, Java, MySQL).
- Creación y configuración de la base de datos.
- Implementación de la aplicación para importar los datos de una red a la BBDD (Base de datos).
- Implementación del Servidor de la red social.
- Implementación de la parte Cliente.
- Implementación del protocolo.
- Pruebas finales.

Comentar además que la evolución de la memoria y de la aplicación ha sido de forma paralela, así cuando se lograban los resultados deseados se incluía en la memoria el desarrollo correspondiente. De esta forma se ha intentado no producir retrasos en ninguna de las partes que componen el PFC.

1.4 Planificación

La primera tarea realizada en este proyecto ha sido la de elaborar el plan de trabajo. Esta planificación inicial viene descrita en la *Figura 1.1*, donde se indica la duración de las distintas tareas así como las dependencias entre cada una de ellas.



	Nombre de tarea	Duración	Comienzo	Fin
1	**Preparación proyecto**	**8 días**	**mié 02/03/11**	**vie 11/03/11**
2	Elaboración del plan de trabajo	2 días	mié 02/03/11	jue 03/03/11
3	Redacción de la introducción	6 días	vie 04/03/11	vie 11/03/11
4	**Descripción esquema**	**10 días**	**lun 14/03/11**	**vie 25/03/11**
5	Redacción de la descripción del esquema	10 días	lun 14/03/11	vie 25/03/11
6	**Diseño e implementación del esquema**	**25 días**	**lun 28/03/11**	**vie 29/04/11**
7	Diseño	5 días	lun 28/03/11	vie 01/04/11
8	Implementación	20 días	lun 04/04/11	vie 29/04/11
9	Redacción del diseño y la implementación	25 días	lun 28/03/11	vie 29/04/11
10	**Pruebas**	**20 días**	**lun 02/05/11**	**vie 27/05/11**
11	Diseño del juego de pruebas	5 días	lun 02/05/11	vie 06/05/11
12	Realización de las pruebas	10 días	lun 09/05/11	vie 20/05/11
13	Estudio de los resultados	5 días	lun 23/05/11	vie 27/05/11
14	Redacción de las pruebas y del estudio	20 días	lun 02/05/11	vie 27/05/11
15	**Adecuaciones finales**	**5 días**	**lun 30/05/11**	**vie 03/06/11**
16	Preparación del código a entregar	1 día	lun 30/05/11	lun 30/05/11
17	Redacción de las conclusiones	2 días	mar 31/05/11	mié 01/06/11
18	Revisión final de la memoria	2 días	jue 02/06/11	vie 03/06/11
19	**Entrega del PFC**	**1 día**	**lun 06/06/11**	**lun 06/06/11**
20	Presentación de la defensa del PFC	1 día	lun 06/06/11	lun 06/06/11

Figura 1.1 – Planificación del proyecto

1.5 Productos obtenidos

Una vez terminado el desarrollo, el sistema consta de las siguientes clases: *Client*, *Crypto*, *Protocol*, *Query*, *Response*, *Server*, *ServerThread*, *User*, *WSE* y *XML*. Además se incluyen las clases *Statistics* y *Test* que sirven para guardar estadísticas y realizar pruebas sobre el protocolo respectivamente.

Además se ha implementado una aplicación para poder importar los datos contenidos en un fichero a la base de datos. Estos datos contenidos en el fichero representan las relaciones de amistad entre los usuarios de una red social. Las clases que componen esta aplicación son *ReadGraphGUI*, *ReadGraph* y *DLfilter*.

1.6 Estructura de la memoria

En los siguientes capítulos de la memoria se describe la teoría del protocolo, además de sus fases de diseño, implementación y pruebas. A continuación se describe el contenido de los capítulos de la memoria:

- Descripción del esquema. En el capítulo 2 se detalla la arquitectura y el funcionamiento del protocolo.
- Diseño e implementación. En el capítulo 3 se justifican las decisiones de diseño que se han tomado y se detallan los componentes del esquema. Además se describen brevemente todas las clases de la aplicación.
- Pruebas. El capítulo 4 explica la metodología seguida para realizar las pruebas del protocolo. Se describe además la aplicación *ReadGraph* que se ha implementado para importar una red social a la base de datos y de esta forma poder realizar las pruebas pertinentes del protocolo.
- Resultados de las simulaciones. El capítulo 5 presenta los resultados de cada una de las pruebas realizadas sobre el protocolo. .
- Conclusiones. En el capítulo 6 se evalúan los objetivos del proyecto, se explican los aspectos más relevantes en el desarrollo del proyecto y se explican posibles mejoras que se pueden realizar del proyecto en el futuro.

- Bibliografía. En la bibliografía, que corresponde al capítulo 7, se pueden encontrar las referencias a toda la documentación que se ha consultado durante el desarrollo del proyecto.
- Anexos. En los capítulos 8 y 9 se detallan los pasos a seguir para instalar y poner en marcha el sistema para poder realizar pruebas sobre el protocolo.

Capítulo 2

Descripción del esquema

En este capítulo se detalla el funcionamiento del esquema propuesto. Concretamente, en el apartado 2.1 se describe la arquitectura del sistema, y a continuación en la sección 2.2 se enumeran los tipos de usuarios que se pueden encontrar en el protocolo, que se describe detalladamente en la sección 2.3. Para terminar, en la sección 2.4 se profundiza en la privacidad que proporciona el protocolo, explicando la firma digital y su integración en el sistema.

2.1 Arquitectura

El esquema planteado se basa en el funcionamiento del protocolo propuesto en [2]. Se encuentran dos entidades participantes en el esquema:

- Usuarios. Son los encargados de enviar sus consultas al buscador. Su principal preocupación es la de mantener su privacidad a salvo de los motores de búsqueda.
- Motor de búsqueda (World Search Engine, WSE). Construyen perfiles de sus usuarios con el fin de ofrecerles un mejor servicio, por lo que su privacidad se ve expuesta. Además, también pueden comercializar sus perfiles con entidades externas. Por lo tanto, los WSE no están interesados en proteger la privacidad de sus clientes.

El sistema implementado para proteger la privacidad de los usuarios requiere que éstos hagan uso de las redes sociales. Las redes sociales permiten mantener conectados usuarios con intereses comunes, hecho que aprovecha el sistema para poder distribuir las consultas de un usuario entre los usuarios del grupo de la red social, permitiendo al WSE obtener un perfil de cada usuario, aunque en

realidad sea uno distorsionado. Además, al compartir intereses, el perfil creado por el WSE puede ser útil para ofrecerle un buen servicio al usuario.

El sistema diseñado en este proyecto se basa en redes sociales que permiten conocer el número de amigos que tiene un amigo de un usuario. Esto es de utilidad para poder distribuir equitativamente todas las consultas a través de la red.

Una descripción breve del funcionamiento del protocolo sería la siguiente: un usuario *U* genera consultas que puede enviar directamente al WSE o puede enviarlas a un vecino (un vecino es una relación directa en la red social). Un vecino que recibe una consulta puede enviarla directamente al WSE o puede reenviarla a uno de sus vecinos. Este proceso se repite hasta que alguien envía la consulta al WSE. El perfil de *U* es distorsionado por las consultas que envía al WSE pero que son generadas por otros usuarios de la red social.

2.2 Tipos de usuarios

En un entorno ideal, los usuarios deberían actuar correctamente. Sin embargo, esto no se puede garantizar en un entorno real. Por lo que se pueden definir dos tipos de usuarios que se pueden encontrar en el sistema:

- Usuario honesto. Es aquel que cumple el protocolo propuesto. Esto es, un usuario decide si enviar una consulta directamente al WSE o a uno de sus vecinos, que la enviará en su lugar. Cuando un usuario honesto recibe una consulta de uno de sus vecinos, decide si la envía él al WSE o si la reenvía a otro vecino.
- Usuario egoísta. Es aquel que no sigue el protocolo propuesto. Es decir, cuando desea enviar una consulta al WSE, la envía directamente a un vecino para que la envíe en su lugar. Además, cuando recibe una consulta de otro usuario la descarta y no da ninguna respuesta.

Los usuarios egoístas impiden que las consultas sean distribuidas uniformemente a través del grupo de la red social. Esta situación puede poner en riesgo la

privacidad de los usuarios honestos, por lo que el sistema propuesto utiliza un mecanismo que previene a los usuarios de comportarse de esta forma, y que se describe en la *Sección 2.3.4*.

2.3 Descripción del protocolo

Sea un usuario U_i miembro de una red social, con un número determinado k de vecinos (relaciones directas en la red social) $\{N_1, \ldots, N_k\}$. Como se ha mencionado anteriormente, la red social permite a U_i conocer el número de vecinos (amigos) que tiene cada uno de sus vecinos. Con esta información, U_i puede calcular la probabilidad de envío P_s para cada uno de sus vecinos (descrita en la *Sección 2.3.1*). Además de estos datos, U_i mantiene una medida del nivel de egoísmo de cada uno de sus vecinos.

Un usuario U_i que desee enviar una consulta al WSE seguirá el siguiente protocolo:

1. U_i debe decidir quién envía la consulta al WSE, si él o uno de sus vecinos $\{N_1, \ldots, N_k\}$. Para ello ejecutará la *función de selección de usuario* Ψ, que se explica en la *Sección 2.3.2*. En este punto se pueden llevar a cabo dos acciones dependiendo del usuario devuelto por la función Ψ:
   a) Si la función Ψ determina que es U_i el que debe enviar la consulta, entonces U_i envía directamente la consulta al WSE.
   b) Si la función Ψ devuelve un vecino N_j, entonces U_i le envía la consulta a dicho vecino. Aquí dependiendo de si N_j acepta o rechaza la consulta q, se encuentran dos posibles acciones:
      - Si N_j rechaza q, entonces U_i penaliza a dicho vecino actualizando negativamente el parámetro que mide el nivel de egoísmo de N_j.
      - Si N_j acepta q, entonces U_i inicializa un temporizador. Si la respuesta de N_j llega antes de que finalice el temporizador, U_i actualizará positivamente el parámetro que mide el nivel de egoísmo de N_j. En otro caso, U_i actualizará negativamente dicho parámetro.

Este proceso se repite hasta que alguien acepte *q*. En caso de que ningún vecino acepte la consulta, U_i envía la consulta *q* él mismo al WSE.

2. N_j ejecuta la función de egoísmo $\acute{Y}(U_i)$, que determinará si aceptar o no la consulta *q* recibida de U_i. De esta forma se pueden dar dos casos:
   a) Si N_j acepta *q*, éste se convierte en el nuevo responsable de tratar la consulta. En este punto repite el primer paso del protocolo reemplazando la función Ψ por la función Ψ_f para determinar si debe enviar la consulta al WSE o a uno de sus vecinos. El funcionamiento de esta función se detalla en la *Sección 2.3.4*.
   b) Si N_j rechaza *q*, entonces N_j finaliza su participación en el protocolo y U_i actualizará negativamente el nivel de egoísmo de N_j.

2.3.1 Probabilidad de envío P_s

Con el fin de distribuir equitativamente las consultas a través de la red social, cada usuario asigna una probabilidad de envío P_s a cada uno de sus vecinos. La probabilidad de que un usuario U_i envíe una consulta a uno de sus vecinos N_j es proporcional al número de vecinos de N_j. De esta forma, definiendo U_i como un usuario que tiene como grupo de vecinos a $\{N_1, \ldots, N_k\}$, y $\{H_1, \ldots, H_k\}$ el número de vecinos que tiene cada uno de los vecinos de U_i (H_j es el número de vecinos de N_j), se puede calcular la probabilidad de envío de un usuario U_i a un vecino N_j como:

$$P_s(N_j) = \frac{H_j}{\sum_{f=1}^{k} H_f + 1}$$

2.3.2 Función de selección de usuario Ψ

Un usuario que desea enviar una consulta, debe ejecutar la función Ψ con el objetivo de conocer quién es el responsable de enviar la consulta al WSE. Así, un usuario U_i que desea enviar una consulta q, y que tiene la siguiente lista de vecinos $\{N_1, \ldots, N_k\}$, ejecuta la función Ψ para decidir qué usuario de entre $\{U_i,N_1,\ldots, N_k\}$ debe enviar la consulta al WSE.

Tal y como se explica en [2], el sistema utiliza la probabilidad P_s para distribuir equitativamente las consultas en una distancia de longitud dos, es decir, un usuario envía al WSE el mismo número de consultas que sus vecinos y que los vecinos de sus vecinos. Esto es posible ya que un usuario tiene información sobre el número de vecinos de cada uno de sus vecinos, sin embargo no es posible controlar la distribución de las consultas en distancias de longitud superior a dos.

Este hecho, permite que en un caso extremo en el que cada miembro del grupo siempre envíe la misma consulta, el WSE pueda conocer la estructura del grupo y además ser capaz de determinar las relaciones directas e indirectas (en un camino de longitud dos) de cualquier usuario dentro del grupo. Con esta información le resulta sencillo al WSE calcular el centroide de los usuarios que han enviado el mismo número de consultas y determinar el origen de las mismas.

Esta debilidad se soluciona modificando P_s para ofuscar los usuarios en $\{U_i,N_1, \ldots, N_k\}$ añadiendo vecinos más distantes. De esta forma, el sistema crea un vector que contiene los usuarios $\{U_i,N_1, \ldots, N_k\}$, en el que cada uno aparecerá repetido tantas veces como número de vecinos tenga, excepto el usuario U_i que sólo aparecerá una vez. Este vector se repetirá un número aleatorio de veces, y posteriormente se eliminará un intervalo del vector al azar. Este vector final es el nuevo P_s.

El intervalo eliminado del vector, denominado por Θ, tiene la función de crear varianza en el sistema, con el objetivo principal de evitar que el WSE pueda calcular un centroide de grupo y por consiguiente que no pueda identificar el

origen de la consulta. Este intervalo, seleccionado entre dos porcentajes, ha sido calculado empíricamente y está justificado en [2]. Las simulaciones fueron realizadas ejecutando 1000 tests usando redes sociales de 1000 usuarios. Los resultados de las simulaciones llevadas a cabo en el documento mencionado se pueden observar en la *Tabla 2.1*, que muestra la posición media y la desviación de los usuarios con 10 vecinos para diferentes intervalos Θ.

Intervalo Θ	Posición media	Varianza	Desviación
0% - 0%	5,81	8,34	2,89
80% - 40%	4,36	11,54	3,39
60% - 40%	4,46	11,37	3,37
80% - 20%	4,51	10,71	3,27
80% - 60%	4,51	12,5	3,53
40% - 20%	4,49	10,13	3,18
30% - 10%	4,42	10,12	3,18
30% - 20%	4,46	10,21	3,19
20% - 10%	4,3	9,67	3,11

Tabla 2.1 – Posición media, varianza y desviación obtenidas con diferentes Θ intervalos

Como se puede observar, los datos demuestran que la mayor varianza se obtiene con el intervalo 80% - 60%. Así que éste será el intervalo seleccionado por el sistema para ser eliminado y obtener como resultado el vector final de usuarios.

Para terminar, se seleccionará una posición aleatoria de ese vector final para determinar el usuario que debe procesar la consulta.

2.3.3 Función de reenvío de consulta Ψ_f

Un usuario U_i que recibe una consulta q de uno de sus vecinos N_j debe procesarla para concluir qué hacer con ella. La función Ψ_f determinará si es el propio usuario

U_i el que debe enviar la consulta *q* al WSE o si la debe reenviar a uno de sus vecinos $\{N_1, \ldots, N_k\}$.

La función devolverá el encargado de tratar la consulta. La selección se hará de forma aleatoria de entre los usuarios del vector $\{U_i, N_1, \ldots, N_k\}$, siendo U_i el propio usuario que ejecuta la función y $\{N_1, \ldots, N_k\}$ su lista de vecinos.

De esta forma, si la función devuelve U_i, entonces éste enviará *q* al WSE. Si devuelve N_j, entonces U_i reenviará la consulta *q* a su vecino N_j.

2.3.4 Función del nivel de egoísmo ΄Υ΄

El objetivo principal de esta función es la de penalizar a los usuarios que se comportan de forma egoísta. Cuando un usuario *U* recibe una consulta *q* de uno de sus vecinos *N* ejecuta la función ΄Υ΄ para decidir si aceptarla o no.

Inicialmente un usuario *U* asigna a cada uno de sus vecinos una probabilidad *p* de aceptar consultas que provengan de dicho vecino. Para comenzar, todos los vecinos tienen una probabilidad asignada de *p=1* (100% de probabilidad). Se utilizará la notación $p_{U,N}$ para representar la probabilidad de que *U* acepte una consulta de *N*.

En el momento en el que un usuario *U* recibe una consulta *q* de un usuario *N*, puede pasar lo siguiente:

1. Si *U* acepta *q*, se completarán las siguientes acciones:
   - *N* aumentará su probabilidad de aceptar consultas de *U*:

$$p_{(N,U)} = p_{(N,U)} + (2 \cdot \zeta)$$

Donde ζ es una constante definida en el sistema y cuyo valor se basa en los cálculos empíricos y justificados en [2]. Las simulaciones realizadas demuestran que el valor óptimo para prevenir a los usuarios honestos

de comprometer su privacidad aislando a los usuarios egoístas es para *ζ=0,03*. Las simulaciones han sido realizadas ejecutando 100 test sobre una red social de 1000 usuarios, de los que el 10% fueron configurados para comportarse de forma egoísta y en donde cada usuario envía su propia consulta 100 veces. Los resultados se pueden observar en la *Tabla 2.2*.

ζ	% usuarios honestos expuestos	% usuarios egoístas expuestos
0,00	2,4	2
0,01	2,3	5
0,02	2,2	93
0,03	2,2	100
0,04	2,4	100
0,06	2,56	100
0,08	2,89	100
0,10	2,89	100

Tabla 2.2 – Porcentaje de usuarios expuestos para diferentes valores de ζ

- *U* decrementa su probabilidad de aceptar consultas de *N*:

$$p_{(U,N)} = p_{(U,N)} - \zeta$$

2. Si *U* rechaza *q*, entonces *N* decrementa su propia probabilidad de aceptar las consultas que provengan de *U*:

$$p_{(N,U)} = p_{(N,U)} - \zeta$$

La función de estas operaciones es la de incentivar a los usuarios que acepten consultas de sus vecinos y penalizar a aquellos que las rechacen. De esta forma, si un usuario *U* comienza a rechazar consultas de sus vecinos, la probabilidad de que sus vecinos acepten consultas que

provengan de él decrementará hasta llegar a cero, obligando a U a enviar directamente sus consultas al WSE, y aislándolo de esta forma del sistema.

2.4 Privacidad de los usuarios

El sistema implementado provee anonimato a los usuarios que no están directamente conectados en la red social. Esto es, si una consulta es creada por un usuario U_1, y considerando que sigue el camino $U_1 \rightarrow U_2 \rightarrow U_3 \rightarrow U_4$ antes de ser enviada por el usuario U_4 al WSE, el usuario U_4 sólo podrá saber que la consulta q proviene de U_3, pero no tendrá información de qué usuarios o cuántos hay antes que U_3. Lo mismo ocurre entre U_3 y U_2, y así sucesivamente.

Este poder de anonimato de un usuario puede provocar que un usuario deshonesto envíe una consulta que vulnere la ley. De esta forma, si un usuario U_1 genera una consulta ilegal, y ésta es enviada al WSE por otro usuario U_3, dado que el funcionamiento del sistema es transparente al usuario, U_3 no podrá comprobar que se trata de una consulta con un contenido ilegal. Esto le hará responsable de dicha consulta ante cualquier autoridad gubernamental, aún sabiendo U_3 que él no ha generado dicha consulta.

El esquema diseñado se basa en una medida *a posteriori* utilizada en el protocolo descrito en [3] para solventar este problema. Esta medida trata de la utilización de la firma digital para probar la transacción entre dos usuarios. Las firmas digitales están aprobadas por la Ley Europea y son admisibles como pruebas válidas en procedimientos legales [4].

De esta forma, el sistema provee anonimato a un usuario frente a los demás, pero no frente a una autoridad gubernamental que investigue a un usuario relacionado con una consulta ilegal.

2.4.1 Firma digital

El sistema propuesto utiliza la firma electrónica avanzada que se basa en un certificado reconocido y que es creado por una CA (Autoridad Certificadora). Este tipo de firma digital tiene un fuerte valor judicial, garantizando autenticación, integridad, confidencialidad y no repudio. Su inclusión en el sistema tiene el fin de permitir a un usuario honesto probar ante una autoridad gubernamental que él no ha generado una determinada consulta.

Cabe mencionar que en el protocolo implementado no se incluye la verificación de la clave pública por no disponer de cada usuario de una clave pública certificada por una CA. Sin embargo cada usuario dispone de su par de claves (pública y privada), realizándose de igual forma el cifrado y verificación de consultas en cada reenvío, además de su almacenaje en disco. Estos puntos son clave para poder realizar una estimación del tiempo de proceso de usuario que se consumiría en un entorno real.

2.4.2 Integración en el sistema

Para poder probar la inocencia de un cierto usuario, el esquema propuesto utiliza certificados como prueba de la transacción entre dos usuarios. Por ejemplo, si en un camino $U_1 – U_2 – U_3$, donde U_1 es el creador de la consulta q y U_3 el que la envía al WSE, U_1 envía el par $(q, \sigma_{U1,U2}(q))$ a U_2, siendo $\sigma_{U1,U2}(q)$ el certificado que prueba la transacción entre U_1 y U_2. El certificado es firmado por U_1 utilizando su clave secreta (SK_{U1}). La clave pública de U_1 se asume que es conocida y aceptada por sus vecinos, por lo que U_2 verifica la firma y posteriormente envía $(q, \sigma_{U2,U3}(q))$ a U_3. Se puede ver el proceso gráficamente en la *Figura 2.1*.

El certificado de una transacción donde un usuario U_i envía una consulta a U_j, debe reflejar el usuario que envía la consulta, el que la recibe, la propia consulta, y la fecha y hora del envío, y además firmado con la clave secreta de U_i.

El usuario U_j que recibe el certificado deberá guardarlo durante un período de tiempo de entre 6 meses y 2 años, tal y como lo establece la Directiva Europea 2006/24/EC [5], para poder demostrar su inocencia en caso de investigación legal.

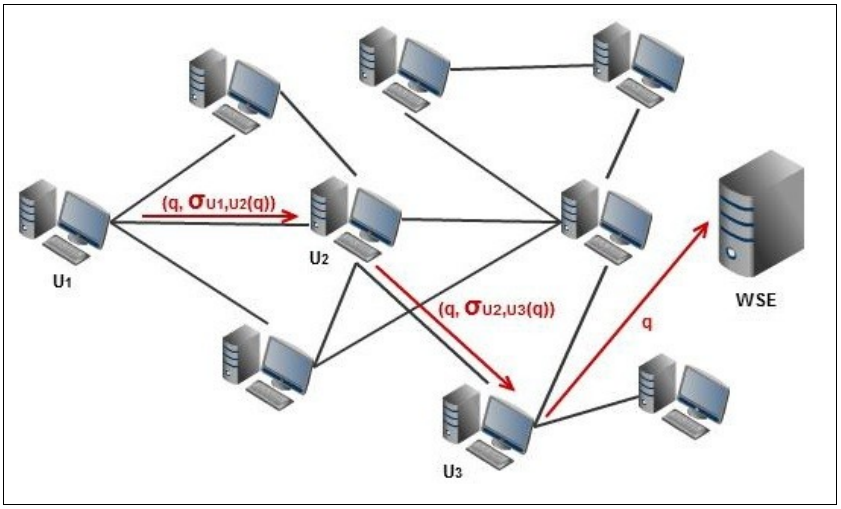


Figura 2.1 – Envío de una consulta q con certificados de transacción

Capítulo 3

Diseño e implementación

Durante el presente capítulo se profundiza en las fases de diseño e implementación del sistema. Concretamente, en el apartado 3.1 se justifican las decisiones tomadas sobre la elección de las herramientas utilizadas para la implementación del protocolo, y seguidamente, en la sección 3.2 se describen los diferentes componentes involucrados en el sistema. El capítulo concluye con el apartado 3.3 mostrando el diagrama de clases de la aplicación y explicando brevemente cada una.

3.1 Decisiones de diseño

En esta sección se describen los criterios que se han tenido en cuenta a la hora de seleccionar las distintas herramientas para la implementación del protocolo.

3.1.1 Lenguaje de programación

Para implementar el protocolo [2] es necesario utilizar un lenguaje que cumpla con las restricciones marcadas por la arquitectura del esquema.

El protocolo hace uso de las relaciones existentes entre los usuarios de las redes sociales. Asimismo, la actual proliferación de dispositivos móviles que permiten el acceso a Internet, y por lo tanto a las redes sociales y a los WSE, permite que se puedan encontrar una gran cantidad de usuarios ejecutando la aplicación desde diferentes máquinas y con diferentes sistemas operativos. Por esta razón, el protocolo debe ser desarrollado utilizando un lenguaje que permita su ejecución en

una gran variedad de dispositivos. Esta restricción obliga a seleccionar un lenguaje de programación que sea multiplataforma.

El protocolo exige que un usuario esté “esperando” a que lleguen consultas de sus vecinos, y cuando llegue una, procesarla mientras se mantiene a la espera de más consultas. Esto obliga a utilizar un lenguaje *multithreaded*, es decir un lenguaje que permita la ejecución simultánea de varios procesos.

Un lenguaje que cumple con estas características es el lenguaje JAVA, que además dadas sus características facilita la implementación de la aplicación que se va a desarrollar:

- *Es simple*, ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. Además posee una curva de aprendizaje muy rápida, debido en parte a su semejanza con C y C++. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.
- *Es orientado a objetos*. Los objetos agrupan estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia actual, en la que se encuentra Java, apunta hacia la programación orientada a objetos, especialmente en entornos cada vez más complejos y basados en red.
- *Es robusto*, realiza verificaciones en busca de problemas en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. También implementa los *arrays auténticos*, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobrescribir o corromper memoria resultado de punteros que señalan a a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

- *Es independiente de la arquitectura*, está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan variopintos, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura, diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.
- *Es multithreaded*, esto quiere decir que Java permite muchas actividades simultáneas en un programa. Hoy en día ya se ven muy limitadas aquellas aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.
- *Es seguro*, impidiendo la suplantación de clases y el acceso ilegal a la memoria.
- *Es dinámico*, Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

3.1.2 Comunicaciones

Ya seleccionado Java como lenguaje de programación, se deben escoger las herramientas disponibles para implementar las comunicaciones de red entre los usuarios del protocolo. Java ofrece dos posibilidades, JXTA [6] y los *Sockets*.

JXTA es un conjunto abierto de protocolos peer-to-peer (P2P) que permiten a cualquier dispositivo en red comunicarse y colaborar mutuamente como iguales. Los protocolos JXTA son independientes del lenguaje de programación y de los

protocolos de transporte. Pueden ser implementados en Java, C/C++, .NET, Ruby y muchos otros lenguajes, y a su vez, pueden ser aplicados sobre TCP/IP, HTTP, bluetooth y otras redes de transporte a la vez manteniendo interoperabilidad mundial.

La red JXTA consiste en una serie de nodos interconectados, en la que cada igual ofrece un conjunto de servicios y recursos que están a disposición de otros iguales (*peers*). Los servicios son programas interactivos y pueden incluir bases de datos, sistemas de autenticación, servidores de chat o cualquier programa que pueda ser conectado. Cada nodo publica sus servicios y recursos usando documentos XML.

En cuanto a los *sockets*, son un sistema de comunicación entre procesos de diferentes máquinas de una red, permitiendo el flujo de datos bidireccional. A través de las clases del paquete *java.net*, los programas Java pueden utilizar los protocolos TCP (Transport Control Protocol) o UDP (User Datagram Protocol) para comunicarse a través de Internet. Algunas de éstas, tales como las clases *Socket* y *ServerSocket* proporcionan un canal de comunicación independiente del sistema utilizando TCP, cada una de las cuales implementa el lado del cliente y el servidor respectivamente.

Para crear una conexión entre un cliente y un servidor, se inicia el servidor utilizando la clase *ServerSocket* indicando el puerto que se utilizará para recibir las peticiones de los posibles clientes. Un cliente que se quiera comunicar con el servidor, deberá utilizar la clase *Socket* e indicar la dirección IP del servidor y su número de puerto. De esta forma, se establece una conexión entre el cliente y el servidor, y que mediante los flujos de datos que proporciona Java permitirá la comunicación entre ambos, estableciendo para ello un canal de lectura y otro de escritura.

Esta simplificación y abstracción de la plataforma utilizada que ofrecen los *Sockets* de Java, permite que la construcción de las comunicaciones entre los distintos usuarios del protocolo sea una tarea más simple y más flexible. Por ello,

se han seleccionado los *Sockets* para implementar las comunicaciones en el entorno de red.

3.2 Diseño de la arquitectura. Componentes

Servidor. Simula el servicio de los usuarios de una red social. Cuando un cliente de la red social se quiere conectar a ésta, se comunica con el servidor para realizar dicha petición. El servidor procesa la petición, cambia el estado del usuario a “conectado” y le envía su lista de vecinos. Además el servidor actualiza la dirección IP del usuario conectado.

Base de datos. Contiene información de los usuarios de la red social, así como la relación de amistad que mantienen entre ellos. Sólo es accesible por el servidor de la red social. Así, dispone de dos tablas:

1. *Tabla de usuarios*. Mantiene la información de cada usuario de la red social: nick, dirección IP, estado y número de amigos.
2. *Tabla de relaciones*. Indica las relaciones que tiene cada uno de los usuarios.

Es necesario comentar que el protocolo implementado será probado en una sola máquina, por lo que la dirección IP de cada usuario será la misma. Para ello se dispondrá de información adicional como es la del puerto de comunicaciones, para poder diferenciar cada uno de los usuarios y poder establecer las conexiones correctamente.

Cliente. Corresponde al usuario de la red social, y a su vez al usuario del protocolo. El cliente se conecta al servidor central de la red social con el fin de conectarse a la misma y poder interactuar con sus amigos. El servidor se conecta a la base de datos para obtener su lista de amigos junto con sus direcciones IP y enviarle la información al usuario. De esta forma el cliente ya podrá iniciar el protocolo, en el que se encargará de:

1. Aceptar o rechazar consultas procedentes de otros usuarios.
2. Procesar las consultas aceptadas, así como las creadas por él mismo.

En la *Figura 3.1* se puede observar la interacción entre los distintos componentes .

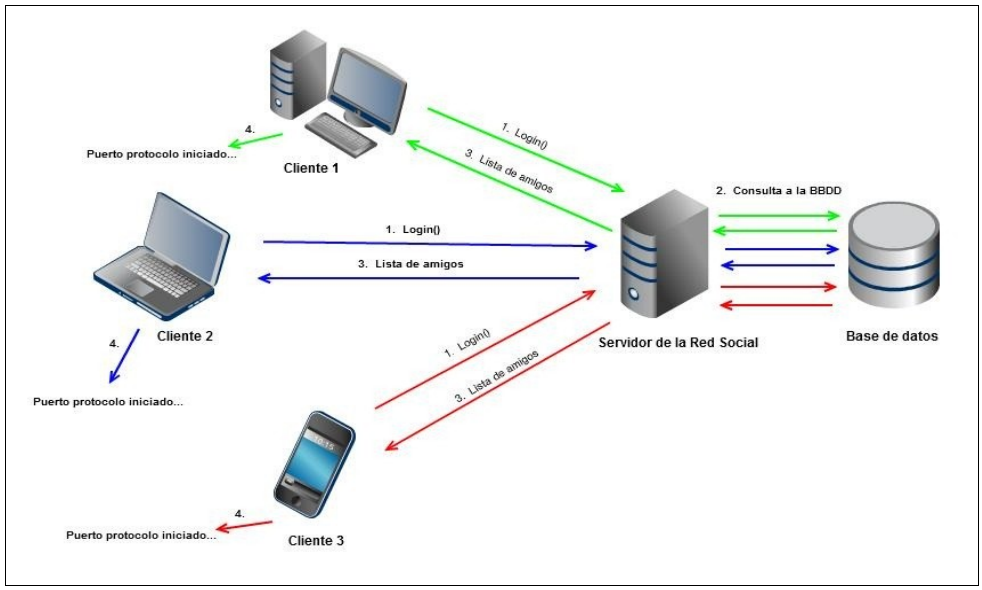


Figura 3.1 – Esquema de la interacción entre componentes

3.3 Diagrama de clases

La *Figura 3.2* corresponde con el diagrama de clases de la aplicación. Se han detallado los atributos y los métodos más significativos de cada una de las clases.

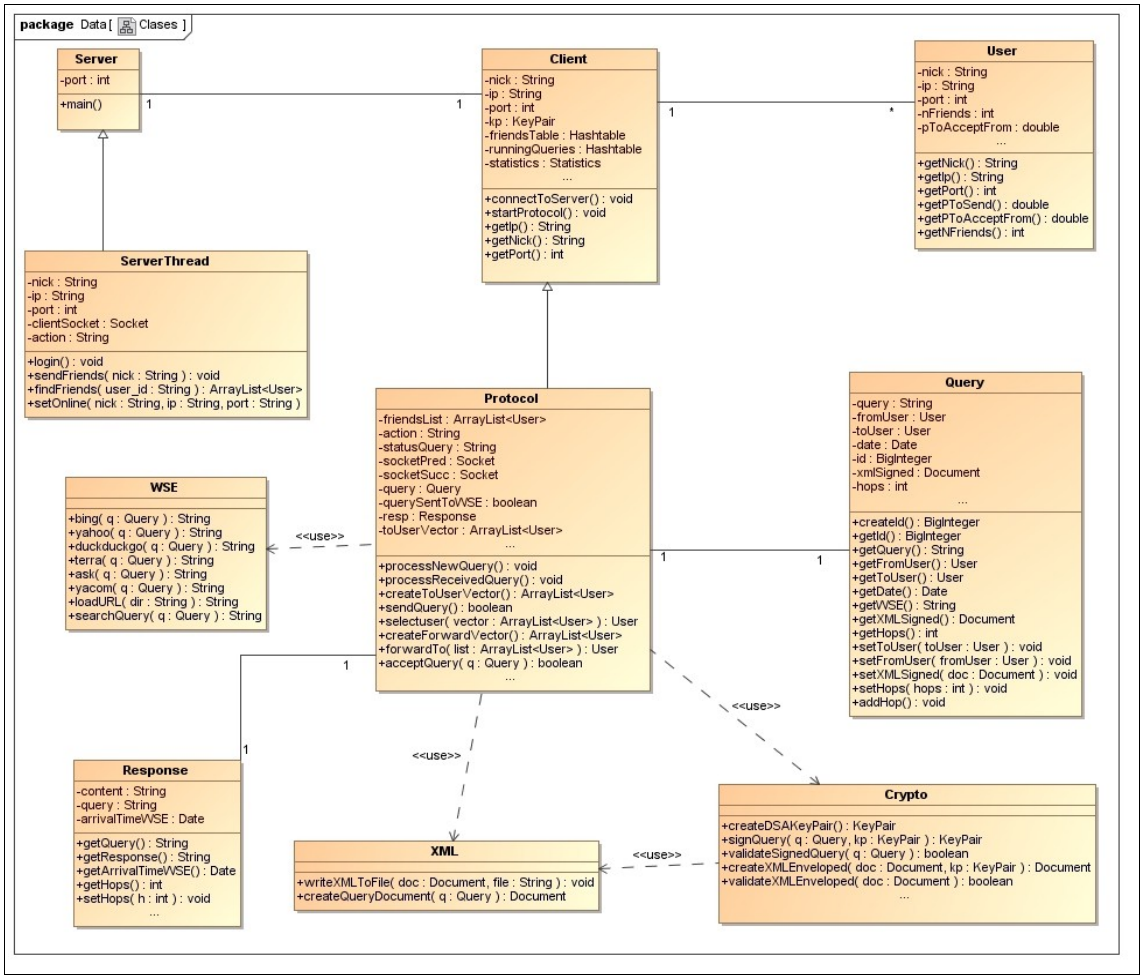


Figura 3.2 – Diagrama de clases

A continuación se resumirá brevemente el funcionamiento de cada clase de la aplicación.

3.3.1 Server

Esta clase simula el comportamiento del servidor de la red social. Dado que en el proyecto actual el objetivo principal es la implementación del protocolo, y posteriormente probar su funcionamiento, el comportamiento del servidor se limita a tratar las peticiones de conexión de los usuarios de la red social, con el fin de

que todos los usuarios estén online y se pueda observar el funcionamiento del protocolo.

El servidor se basa en un socket TCP multihilo. Cuando un cliente de la red social envía una petición de conexión, el servidor la procesa, consulta la base de datos de la red social para elaborar su lista de amigos y se la envía al usuario que ha realizado la conexión. Además actualiza en la base de datos el estado del usuario como conectado.

La clase servidor sigue el siguiente esquema:

- Inicializa un *ServerSocket* (socket de tipo servidor) que se mantendrá en un bucle escuchando peticiones que lleguen al puerto indicado, que será el 12300.
- El servidor sólo procesará las peticiones de conexión de los usuarios.
- Cuando se recibe una petición al puerto del servidor, se procesa para determinar de qué tipo de petición se trata. En este caso sólo puede tratarse de una petición de conexión (LOGIN).
- Al tratarse de un petición de conexión, el servidor inicia un nuevo hilo de ejecución para procesarla. Este hilo pertenece a la clase *ServerThread*, que será explicada posteriormente.
- Como el proceso de la conexión se ejecuta en un hilo aparte, el servidor vuelve a ponerse en escucha por el puerto determinado a la espera de más peticiones.

3.3.2 ServerThread

Esta clase corresponde al hilo que inicia la clase *Server* para procesar una determinada petición. En este caso se tratará de una petición de conexión, por lo que esta clase lo que hace es lo siguiente:

- Establece un diálogo con el usuario que ha realizado la petición para determinar su nombre de usuario (*nick*) y su dirección IP. Además, como el protocolo será ejecutado en local, para poder simular el comportamiento de

varias máquinas el usuario deberá enviarle también un número de puerto que lo diferencie de los demás.

- Actualiza el estado del usuario a conectado.
- Elabora la lista de amigos del usuario conectado y se la envía.

3.3.3 Client

Esta clase implementa la conexión de un usuario a la red social y su puesta en marcha para la ejecución del protocolo. Para que un cliente pueda ejecutar el protocolo debidamente, debe ser capaz de procesar peticiones que procedan de otros usuarios, así como las que genere él mismo. Para ello, la clase sigue los siguientes pasos:

- Envía una petición de conexión al servidor de la red social.
- Recibe su lista de amigos por parte del servidor.
- A partir de la lista recibida, todos los amigos son introducidos en una estructura *Hashtable* (tabla de acceso aleatorio). Esta estructura de Java almacena el par clave/valor de un elemento, para que posteriormente se pueda obtener el valor del mismo a través de su clave. En este caso almacenará los amigos (cada uno representado por la clase *User*), y la clave será el *Nick* de cada uno.
- Inicia el protocolo:
  - Genera su par de claves (pública y privada) para poder firmar las transacciones realizadas con sus vecinos.
  - Crea una directorio en disco en donde se guardarán las transacciones firmadas que realice en el protocolo.
  - Utiliza un *ServerSocket* para abrir una conexión de tipo servidor en el puerto determinado para el protocolo (que al realizarse en local será diferente para cada usuario con el fin de poder diferenciarlos) y mantenerse a la espera de recibir peticiones pertenecientes al protocolo. Esta conexión se mantiene dentro de un bucle “while”.

- Cuando un usuario recibe una petición en el puerto destinado al protocolo, determina de qué tipo de petición se trata:
  - Si es del tipo CREATE_QUERY se trata de una consulta que ha sido generada por el propio usuario, así que deberá procesarla según determina el protocolo. Para ello crea un nuevo hilo en donde procesará dicha consulta y se vuelve a poner en "escucha" en el puerto especificado por si recibe nuevas consultas a procesar. Este nuevo hilo está implementado por la clase *Protocol* que se explicará más adelante.
  - Si es del tipo SEND_QUERY, la consulta proviene de alguno de sus vecinos, por lo que ejecutará la acción a realizar en un nuevo hilo (clase *Protocol*), y procesará la consulta siguiendo el protocolo. Un usuario puede participar en el proceso de varias consultas simultáneamente, por lo que se puede llegar a dar la siguiente circunstancia: Un usuario U_1 recibe una consulta q_1 de su vecino N_1, y una consulta q_2 de su vecino N_2. Luego vuelve a recibir para procesar la misma consulta q_1 pero ahora del vecino N_3. Esta consulta debe ser rechazada por el usuario U_1 debido a que ya la ha tratado y aún no ha recibido la respuesta. En este caso no se aplicará al usuario U_1 la "*función del nivel de egoísmo*" que marca el protocolo.

3.3.4 Protocol

Implementa el funcionamiento del esquema diseñado y hereda de la clase *Client*. Sus tareas son las siguientes:

- Cuando la clase es creada por su clase padre (Client) determina el tipo de acción que debe procesar (SEND_QUERY o CREATE_QUERY). Si es del tipo SEND_QUERY realiza las siguientes funciones:
  1. Ejecuta una función para determinar si acepta o no la consulta, informando en cada caso al vecino que le haya enviado la consulta.

2. En caso de aceptarla, añade dicha consulta en la tabla de “Consultas en proceso” de su clase padre y la guarda firmada en disco.
3. Actualiza la probabilidad de recibir consultas de su vecino según determina el protocolo.
4. Posteriormente ejecuta la función de reenvío para determinar el usuario al que deberá enviar la consulta.
5. En caso de que tenga que enviar la consulta él mismo al buscador, obtiene la respuesta y se la envía al usuario que le había enviado la consulta. En caso contrario, firma la consulta con su clave privada (contenida en la clase *Client*) y la envía al usuario seleccionado.
6. Si el usuario seleccionado al que se le ha enviado la consulta no la acepta, se vuelven a ejecutar los pasos 4, 5 y 6. Además se actualiza la probabilidad de enviarle consultas al usuario que la ha rechazado tal y como marca el protocolo.
7. Una vez enviada la consulta a algún vecino que la acepte, se actualiza su probabilidad de envío a dicho vecino y se mantiene en espera hasta recibir la respuesta del mismo.
8. Una vez recibida la respuesta, la envía al usuario que previamente se la había enviado a él.

- Si el tipo de petición es CREATE_QUERY se realizan las siguientes tareas:
  1. Se ejecuta “la función de selección de usuario” con el fin de determinar el usuario al que se debe enviar la consulta.
  2. Se siguen los pasos 5, 6, 7 y 8 indicados para el tipo de petición SEND_QUERY.

3.3.5 Query

Describe una consulta que es procesada siguiendo el protocolo. Cuando se crea una nueva instancia de esta clase, el constructor de la misma crea un identificador de 128 bits que la diferenciará del resto de consultas. De esta forma un usuario puede rechazar una consulta que recibe si comprueba que su identificador coincide con alguno de las consultas que mantiene en su tabla de consultas en

proceso (es decir, un usuario no podrá procesar más de una vez una misma consulta).

Además mantiene información del usuario del que procede la consulta, del usuario receptor, de la fecha de creación y del buscador al que finalmente será enviada. También guarda un documento XML que representa la consulta además de contener la firma de la misma. Un ejemplo del esquema que sigue este XML se puede ver en la *Figura 3.3. El documento XML sigue la especificación "XML Signature Syntax and Processing (Second Edition)" [7], utilizando para ello la API Java™ XML Digital Signature (JSR 105) [8].*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<query>
<search>cooches usados </search>
<from><nick>user3</nick><ip>localhost</ip><port>12345</port></from>
<to><nick>user2</nick><ip>localhost</ip><port>12346</port></to>
<date>Thu Apr 28 19:55:03 CEST 2011</date>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo><CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <Reference URI="">
            <Transforms><Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/></Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>qHnIii/BD7Ebdzl7XLAgEtuowUY=</DigestValue>
        </Reference>
    </SignedInfo>
    <SignatureValue>lPLIaah9ooWKD9019Hiuusp6XK+IFbMXggbPmLjdQmJNBN4ekBEQsA==</SignatureValue>
    <KeyInfo>
        <KeyValue>
            <DSAKeyValue><P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9
xD7nN1kuFw==</P><Q>l17dzDacuo67Jg7mtqEm2TRuOMU=</Q><G>Z4Rxsnqc9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0Nogps
QW5QvnlMpA==</G><Y>95JtdOzsgCQ8EyCDuqi4jH0QyY6IK3Mqzrk61RzWfEerpVNMFuRz60evckCo7g2gNd+SoJ4wvtQn
GrCLe3M9mA==</Y>
            </DSAKeyValue>
        </KeyValue>
    </KeyInfo>
</Signature>
</query>
```

Figura 3.3 – Esquema XML de una consulta firmada

3.3.6 User

Esta clase es utilizada para representar e identificar a cada usuario, manteniendo la información necesaria para poder ejecutar el protocolo apropiadamente. Los datos que guarda de un usuario son los siguientes:

- Nick.
- Dirección IP.
- Puerto.
- Número de vecinos.
- Las probabilidades de recibir y enviar consultas.

3.3.7 WSE

Implementa las funciones para poder enviar una consulta a un determinado motor de búsqueda de Internet y obtener la correspondiente respuesta.

3.3.8 Response

Representa la respuesta del WSE a una determinada consulta. Guarda en una variable de tipo "*String*" la información de la búsqueda que se ha realizado y en otro "*String*" la respuesta ofrecida por el buscador.

3.3.9 XML

En esta clase se encuentran las funciones relacionadas con la creación y la manipulación de ficheros XML. Concretamente se encuentra la función que crea un documento XML con la información de una consulta específica, y la función que guarda el contenido de un documento XML en un fichero en disco.

3.3.10 Crypto

Incluye los métodos relacionados con la firma digital de una consulta. Las funciones más importantes que se pueden encontrar en esta clase son las siguientes: :

- Creación de una clave privada y su correspondiente clave pública.
- Firma de una consulta con la clave privada de un determinado usuario.
- Validación de una consulta firmada.
- Creación de un documento XML firmado del tipo *XML Enveloped*, que es aquel en el que la firma va embebida en el propio documento XML que se ha firmado.

Capítulo 4

Pruebas

El capítulo actual se divide en tres apartados. El apartado 4.1 describe las redes sociales empleadas en los juegos de pruebas del sistema. A continuación, la sección 4.2 explica la metodología seguida en cada una de las pruebas realizadas. Y finalmente, el apartado 4.3 describe la aplicación desarrollada para realizar la importación de los datos de una red social en la base de datos.

4.1 Red social utilizada

Para realizar los juegos de pruebas, se ha empleado un esquema de relaciones entre usuarios extraído de la red social *Twitter*. Las relaciones que se establecen entre los usuarios y que describen la red social utilizada han sido extraídas y almacenadas en un fichero con el formato DL (Data Language file). Se trata de un fichero de texto que incluye las conexiones entre cada uno de los nodos que conforman el grafo. Este fichero se puede abrir con algún programa de análisis de grafos, como por ejemplo el programa "*visone-2.6.3*", para poder ver gráficamente la relación entre todos los nodos. En la *Figura 3.4* se puede observar el grafo que representa una red social de *Twitter* de 20 usuarios.

Además, en el juego de pruebas se ha utilizado una red social de 100 usuarios para poder analizar las diferencias en el funcionamiento del protocolo en ambas redes. Ésta ha sido creada de forma manual a partir de la red de 20 usuarios extraída de Twitter.

Hay que mencionar que tal y como se puede observar en la *Figura 3.4*, las relaciones entre usuarios en una red social de *Twitter* pueden ser bidireccionales. Esto es debido a que *Twitter* se diferencia de otras redes como *Facebook* o

Tuenti, en que un usuario U_1 puede ser seguidor de un usuario U_2, pero U_2 puede no ser seguidor de U_1. De esta forma, para representar el grafo en la base de datos y posteriormente realizar las pruebas del protocolo, se considerará una relación de amistad entre dos usuarios aunque sólo exista una conexión unidireccional.

Para representar en la base de datos del servidor de la red social el esquema definido en el fichero, se utiliza la aplicación *ReadGraph* implementada para tal efecto, y que se detalla en la *sección 4.3*.

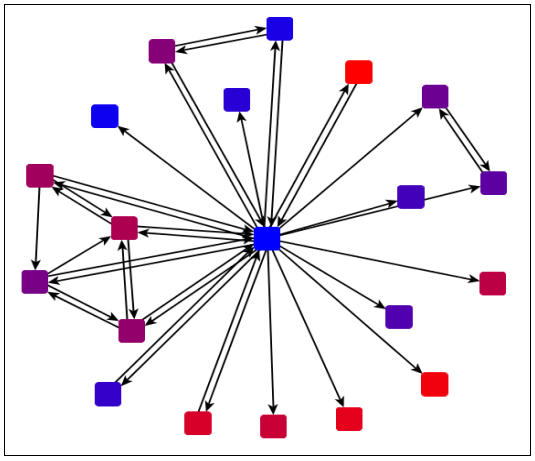


Figura 3.4 – Grafo que representa las relaciones entre los usuarios de la red social Twitter

Tras ordenar a la aplicación *ReadGraph* que cargue los datos del fichero que representa la red social de *Twitter* de 20 usuarios, se puede observar en la *Figura 3.5* como quedan nombrados los diferentes usuarios y las relaciones de amistad existentes entre ellos.

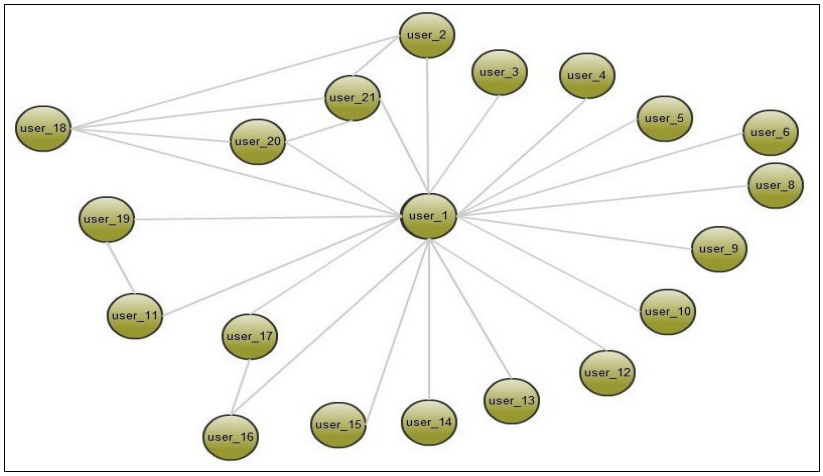


Figura 3.5 - Red de 20 usuarios utilizada para las pruebas del protocolo

La *Figura 3.6* muestra un gráfico que representa el número de amigos de cada usuario de la red social de 20 usuarios.

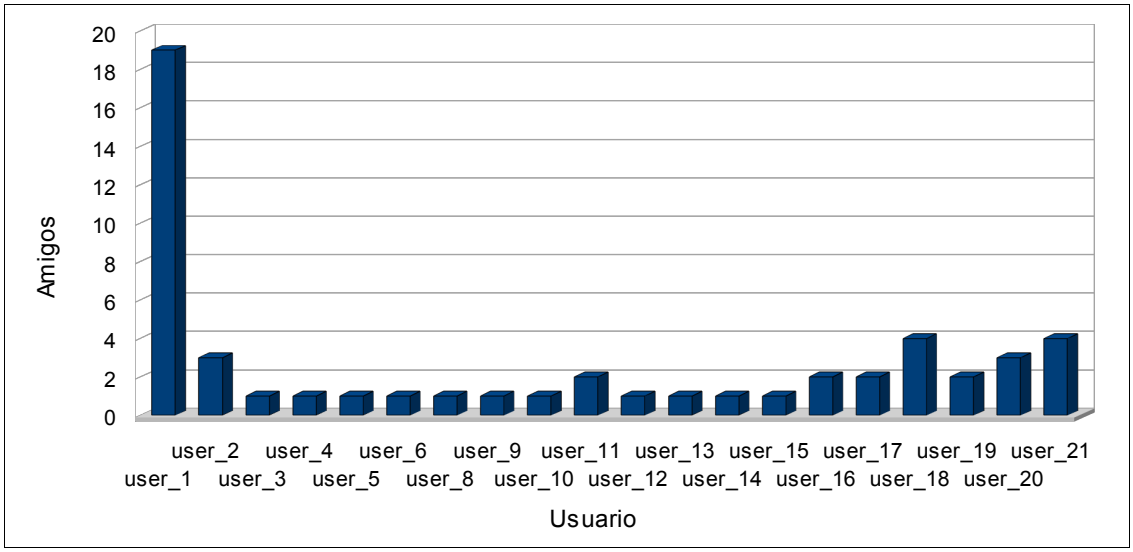


Figura 3.6 – Número de amigos de los usuarios de la red social

4.2 Metodología

Antes de realizar las simulaciones del protocolo, se han ejecutado diversas pruebas, no mostradas en este proyecto, para comprobar el correcto funcionamiento del esquema implementando. Estas pruebas han servido para corregir diversos errores que afectaban al funcionamiento de la aplicación, así como problemas que no se habían considerado previamente en el diseño del esquema. Por ejemplo, la posibilidad de que un nodo recibiese la misma consulta de dos vecinos diferentes (un usuario U_1 envía la consulta q a su vecino U_2, éste a su vez la envía a su vecino U_6 , y éste que tiene como amigo a U_1 le reenvía la consulta). Esto implicaba que un usuario podía participar más de una vez en el proceso de una misma consulta, por lo que se optó a rediseñar esta parte e impedir que un usuario procesase la misma consulta más de una vez y de esta forma evitar la redundancia.

Tras comprobar el correcto funcionamiento del sistema, se han preparado una serie de pruebas para poder evaluar el protocolo. Todos los test han sido ejecutados utilizando consultas reales, extraídas de los ficheros de AOL [9]. Estas consultas han sido descargadas [10], y se ha creado un fichero con unas 650.000 consultas. Antes de realizar los test se lee el fichero de consultas para elaborar una lista en memoria, y de esta forma cada usuario que deba enviar una consulta, la selecciona aleatoriamente de dicha lista.

Las simulaciones se han realizado en una sola máquina, cuyas características hardware y software son las siguientes:

- Procesador: Intel Core 2 Duo T7500 2'2 GHz
- Memoria RAM: 2Gb
- Arquitectura: 64 bits
- Sistema Operativo: Windows 7 Ultimate 64 bits

Por todo esto, los tiempos obtenidos en los test son irreales ya que no se tiene en cuenta la latencia consecuente de la distancia entre los usuarios. Sin embargo, se

pueden hacer estimaciones precisas si se tiene en cuenta el tiempo de proceso de cada nodo y el número medio de saltos por consulta (véase *Sección 5* de [3]).

Además, en los test no se realizará el envío de las consultas al WSE, ya que tampoco sería un dato real al estar realizándose todos los envíos desde el mismo punto (en un entorno real la distancia de cada nodo al WSE será diferente, y por lo tanto, el tiempo también lo será.

De todas formas, para poder hacerse una idea del coste en tiempo de una consulta en un entorno real, se han incluido las variables y operaciones necesarias en la implementación para poder obtener los datos que permiten realizar esta estimación, tales como el número medio de saltos por consulta y el tiempo medio de proceso de cada nodo.

Asimismo, se han incluido en el código de la aplicación otras variables con fines estadísticos, que permitirán conocer datos como la distribución de las consultas de un usuario a través de la red o como el comportamiento de las probabilidades de cada usuario.

Para poder realizar estimaciones precisas y por el hecho de utilizar una sola máquina para las pruebas, el envío de las consultas se realiza de forma secuencial: el usuario "*user_1*" envía su primera consulta. Cuando reciba la respuesta, el usuario "*user_2*" enviará su primera consulta. En el momento que reciba la respuesta, es el usuario "*user_3*" el encargado de enviar su consulta. Así hasta llegar al último usuario "*user_21*" que enviará su primera consulta. Tras esto, el usuario "*user_1*" envía su segunda consulta, y luego "*user_2*" hace lo mismo con su segunda consulta. Y así sucesivamente hasta que se complete el número total consultas que debe enviar cada usuario.

Esto permite que el procesador se ocupe únicamente de una consulta a la vez, porque el hecho de estar utilizando hilos permitiría poder procesar múltiples consultas al mismo tiempo, pero en esta situación no se estarían obteniendo resultados reales en cuanto a los tiempos de proceso de cada usuario.

Con respecto a la criptografía de clave pública, a la hora de realizar la implementación del protocolo y posteriormente las pruebas, no se ha considerado la existencia de las Autoridades de Certificación. Así, un usuario que reciba una clave pública de otro usuario no tendrá forma de verificarla y únicamente deberá confiar en que se trata realmente de su clave pública. De todas formas, los datos del proceso de firma y verificación serán similares y serán útiles para poder realizar estimaciones en un escenario real.

4.3 Carga de datos en la base de datos

Para poder ejecutar el protocolo, se deben tener todos los datos almacenados en la base de datos del servidor de la red social. Por ello, se ha implementado una aplicación que es la encargada de leer la información de un fichero que contiene los datos que representan las relaciones de amistad entre los usuarios de una red social, y almacenar los datos en la base de datos del servidor.

En el fichero, tras el encabezado inicial, cada línea representa una relación de amistad entre dos usuarios. Esta relación entre ambos es indicada mediante el identificador de cada usuario. La aplicación aprovecha estos identificadores para asignar a cada usuario un "*Nick*" para la red social. Así una línea del fichero que contenga "1 2", la aplicación lo interpreta como una relación de amistad entre el usuario "*user_1*" y el usuario "*user_2*".

Esta aplicación consta de tres clases que se explicarán a continuación.

4.3.1 ReadGraph

Implementa todas las funciones necesarias para leer un fichero en disco que contiene los datos de un grafo que representa las relaciones entre usuarios de una red social, y extraer la información necesaria para cargar los datos en la base de datos. Los métodos que recoge son:

- *load (file : File) : void* . Lee línea a línea los datos contenidos en el fichero “file”. Cada línea contiene la relación de amistad entre dos nodos, por lo que para cada una de ellas, extrae el identificador de los nodos y llama al método manage, que se explica a continuación. Posteriormente llama al método updateFriends para actualizar en la base de datos el número de amigos que dispone finalmente cada usuario.
- *manage (node1 : String, node2 : String) : void* . Comprueba si los nodos ya existen. Si no existen se agrega cada uno a la base de datos. Además se selecciona un puerto de comunicaciones libre para cada nodo almacenado y que se guardará en el campo “*PORT*” de la tabla “*USERS*” de cada uno. Este puerto será el que permita la ejecución del protocolo y la comunicación entre los usuarios ya que se debe recordar que todos poseerán la misma dirección IP al ser ejecutados en la misma máquina. Si la relación de amistad entre *node1* y *node2* todavía no se ha leído del fichero, se procede a introducir dicha relación en la tabla “*FRIEND*” de la base de datos.
- *updateFriends() : void* . A partir de las relaciones introducidas en la tabla “*FRIEND*”, se actualiza para cada usuario de la tabla “*USER*” el campo “*N_FRIENDS*”, y que determina su número de amigos en la red social.
- *exists(node : String) : boolean* . Determina si ya ha sido tratado con anterioridad un nodo leído del fichero.

4.3.2 ReadGraphGUI

Esta clase contiene las operaciones que implementan la interfaz gráfica de usuario. Corresponde a una ventana principal y a una ventana de selección de fichero, que permitirán al usuario buscar y seleccionar el fichero que contiene los datos de la red social.

4.3.3 DLfilter

Esta clase es utilizada por la interfaz de usuario *ReadGraphGUI*. Concretamente es utilizada por el objeto que representa la ventana de selección de fichero, para

poder obtener un filtro y sólo mostrar aquellos ficheros aceptados por la aplicación *ReadGraph*. Los ficheros aceptados tendrán la extensión "*.dl*".

4.4 Clases "Statistics" y "Test"

Con el fin de poder evaluar el rendimiento del protocolo se ha desarrollado la clase "*Statistics*" que guardará toda la información relativa a un usuario y necesaria para poder efectuar un análisis del funcionamiento del protocolo.

Además, para poder realizar las pruebas del protocolo se ha desarrollado un pequeño *script*. Éste hará que se conecten todos los usuarios de la red social y de esta forma permitir que estén preparados para participar en el protocolo. La clase que contiene esta pequeña aplicación es la clase "*Test*", que permitirá configurar los parámetros de la prueba e iniciar el proceso (ver *Anexo 2*).

Capítulo 5

Resultados de las simulaciones

Este capítulo recopila los diferentes resultados obtenidos en la fase de pruebas. El apartado 5.1 se exponen los resultados correspondientes a las probabilidades finales de aceptar consultas de cada usuario. El apartado 5.2 muestra los resultados obtenidos con respecto a la distribución de consultas por la red social. La sección 5.3 analiza los resultados obtenidos en cada una de las redes sobre el número medio de saltos de una consulta. El capítulo termina con el análisis de los resultados con respecto al tiempo de proceso de un usuario.

5.1 Probabilidades finales

5.1.1 Comportamiento normal

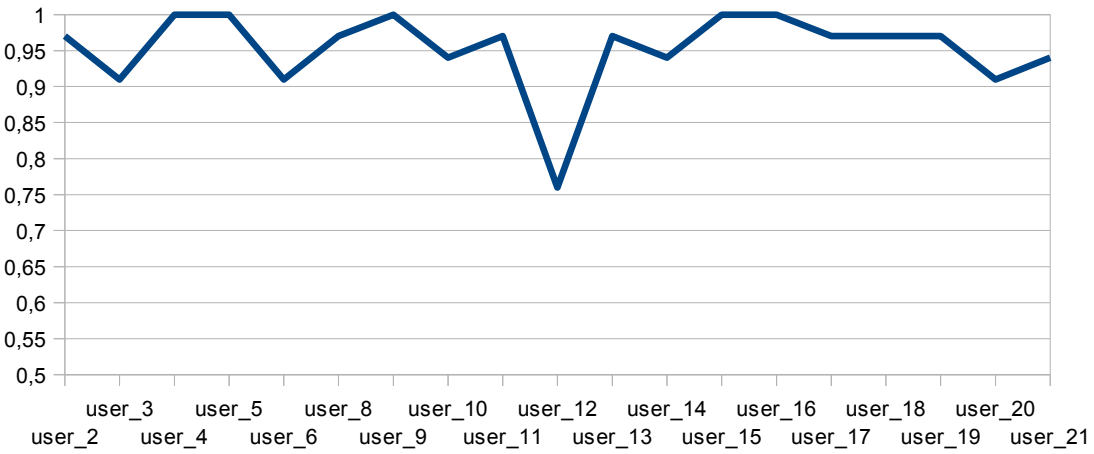
En un primer escenario, se evaluará el funcionamiento del protocolo. Cada usuario seguirá el protocolo y ejecutará 1000 consultas, y al finalizar, se mostrarán los resultados de las probabilidades finales que tiene cada usuario de aceptar consultas de sus respectivos amigos. Recordar, que la probabilidad inicial de un usuario de aceptar una consulta de uno de sus vecinos es de 1. Con los datos obtenidos se podrá observar si queda algún usuario aislado, además de comprobar si la aplicación funciona adecuadamente.

Tras el envío de las 1000 consultas por parte de cada usuario, los gráficos y tablas siguientes muestran para cada uno de ellos, las probabilidades de aceptar consultas de cada uno de sus amigos.

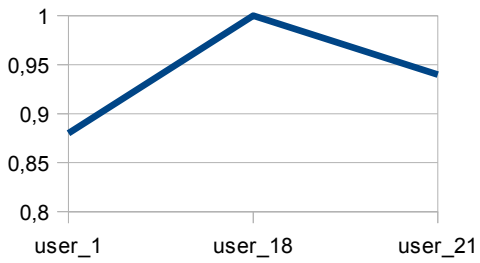
Se observa que todas las probabilidades son muy elevadas, todas están muy cercanas a uno. Esto indica que el protocolo, siguiendo todos los usuarios un

comportamiento normal, funciona correctamente, ya que todos mantienen una alta predisposición a aceptar consultas de sus vecinos.

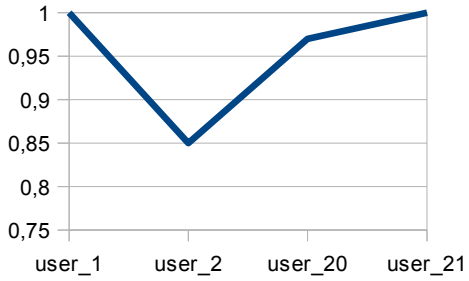
user_1



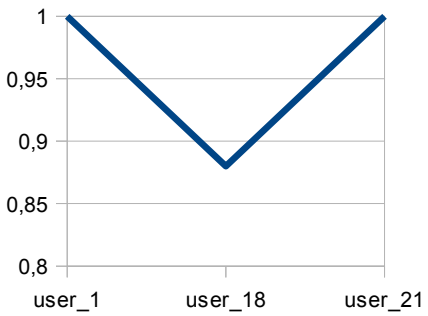
user_2



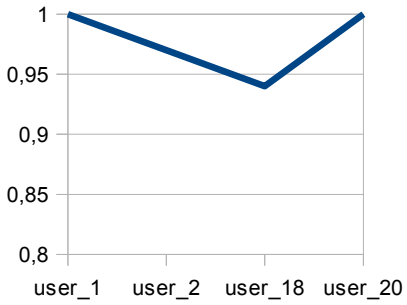
user_18



user_20



user_21



user_3	
user_1	1

user_4	
user_1	0,97

user_5	
user_1	0,94

user_6	
user_1	1

user_8	
user_1	1

user_9	
user_1	0,97

user_10	
user_1	0,97

user_11	
user_1	0,97
user_19	0,94

user_12	
user_1	1

user_13	
user_1	1

user_14	
user_1	1

user_15	
user_1	0,97

user_15	
user_1	0,97

user_16	
user_1	0,94
user_17	0,97

user_17	
user_1	0,97
User_16	0,94

user_19	
user_1	1
user_11	0,94

Tabla 5.1 – Probabilidades finales de los usuarios de la red social

- Tiempo medio por consulta: 0.26088 segundos
- Número medio de saltos por consulta: 2,1152
- Tiempo medio de proceso de cada usuario: 0,12333 segundos

Estos datos también reflejan que ningún usuario queda aislado, es decir, no existe ningún usuario que tenga algún amigo con una probabilidad baja de aceptar consultas que provengan de él, por lo que en principio la privacidad de cada usuario no estaría expuesta. Aunque este hecho se deberá confirmar con la prueba de distribución de consultas (*Sección 5.2*)

5.1.2 Comportamiento egoísta

En el siguiente test, se comprobará el funcionamiento del protocolo en caso de que un usuario se comporte de forma egoísta. En este caso, se ha modificado la implementación para que el usuario "*user_18"* se comporte de esta forma, es decir, rechazando todas las consultas de sus vecinos. Así podremos evaluar si el sistema actúa correctamente y observar si el usuario egoísta es penalizado por el

protocolo aislándolo y obligándolo a enviar sus consultas directamente al WSE. Cada usuario enviará 1000 consultas.

Los resultados se pueden ver en la *Tabla 5.2* y muestran las probabilidades de cada usuario de aceptar consultas de cada uno de sus vecinos.

user_1	
user_2	0,94
user_3	1
user_4	1
user_5	0,97
user_6	0,97
user_8	0,85
user_9	1
user_10	0,94
user_11	0,97
user_12	0,94
user_13	1
user_14	0,94
user_15	0,91
user_16	0,97
user_17	0,9
user_18	0
user_19	1
user_20	0,97
user_21	0,97

user_2	
user_1	0,91
user_18	0
user_21	0,97

user_3	
user_1	0,97

user_4	
user_1	0,91

user_5	
user_1	0,97

user_6	
user_1	1

user_8	
user_1	1

user_9	
user_1	0,91

user_10	
user_1	1

user_11	
user_1	0,97
user_19	1

user_12	
user_1	1

user_13	
user_1	0,97

user_14	
user_1	0,97

user_15	
user_1	1

user_16	
user_1	0,97
user_17	1

user_17	
user_1	1
User_16	0,94

user_18	
user_1	0
user_2	0
user_20	0
user_21	0

user_19	
user_1	0,97
user_11	0,91

user_20	
user_1	1
user_18	0,94
user_21	0

user_21	
user_1	1
user_2	0,97
user_18	0
user_20	0,88

Tabla 5.2 – Probabilidades finales con un comportamiento egoísta

- Tiempo medio por consulta: 0.3457 segundos
- Número medio de saltos por consulta: 1,87375
- Tiempo medio de proceso de cada usuario: 0,12031 segundos

Los resultados muestran claramente, como los usuarios que tienen como vecino al usuario "*user_18*", concluyen con probabilidad cero de aceptar consultas que provengan de él. Estos datos tienen como consecuencia que el protocolo obligue a "*user_18*" a enviar sus propias consultas directamente al WSE, ya que ninguno de sus vecinos aceptará sus consultas, por lo que el protocolo no protegerá su identidad frente al WSE.

Observando las tablas anteriores, que muestran los vecinos de cada usuario, se llegaría a un caso extremo en caso de que el usuario que se comportase de forma egoísta fuese el "*user_1*". Se puede ver como muchos usuarios lo tienen a él como único vecino, por lo que si el "*user_1*" rechazase todas las consultas de sus vecinos los obligaría a enviar directamente todas sus consultas al WSE, por lo que su privacidad podría ser expuesta al WSE.

Reflexionando en este sentido, y sabiendo que un usuario conoce el número de amigos que tiene cada uno de sus vecinos, se puede dar el caso que un usuario U_1 sólo tenga un amigo U_2. De esta forma U_2 que tiene como amigo a U_1 sabe que él es el único amigo de U_1. Así que U_2 tiene claro que todo lo que le envía U_1 es una consulta suya, por lo que podría elaborar un perfil de U_1 y el protocolo no estaría protegiendo su privacidad. Además si U_2 se comporta de forma egoísta, U_1 estaría obligado a enviar sus consultas al WSE por lo que su privacidad también se vería expuesta. Caso que también se repetiría cuando U_2 no esté online.

Con estos datos se llega la conclusión de que para conseguir un mínimo de protección es necesario tener como mínimo dos amigos.

5.2 Distribución de consultas

Se realizarán un par de pruebas que permitan analizar la distribución de las consultas enviadas por cada usuario en dos escenarios distintos. En el primero todos los usuarios seguirán las pautas que marca el protocolo y en el segundo el usuario "*user_18*" se comportará de forma egoísta no aceptando ninguna consulta

de sus vecinos. Así se podrá observar si las consultas de cada usuario son distribuidas de forma más o menos equitativa, y quienes son los responsables de enviar la consulta de un determinado usuario al WSE en ambos escenarios.

Mencionar que estos datos no pertenecen a la implementación propia del protocolo, sino que se han incluido las operaciones necesarias en la programación para poder obtener estos resultados. Es decir, en un entorno real no se podría conocer la distribución de las consultas.

Las tablas siguientes muestran para cada usuario, cuántas de sus consultas han enviado al WSE cada uno de los usuarios de la red.

5.2.1 Comportamiento normal

Consultas enviadas al WSE por usuarios sin relación directa

user_1	
user_1	36
user_2	102
user_3	22
user_4	32
user_5	37
user_6	27
user_8	35
user_9	31
user_10	29
user_11	53
user_12	20
user_13	33
user_14	37
user_15	36
user_16	67
user_17	65
user_18	95
user_19	67
user_20	94
user_21	82

Consultas enviadas al WSE por amigos del usuario

user_2	
user_1	41
user_2	35
user_3	44
user_4	44
user_5	39
user_6	37
user_8	45
user_9	54
user_10	40
user_11	39
user_12	55
user_13	47
user_14	42
user_15	51
user_16	52
user_17	40
user_18	101
user_19	36
user_20	72
user_21	86

Consultas propias enviadas directamente al WSE

user_3	
user_1	48
user_2	52
user_3	76
user_4	48
user_5	48
user_6	51
user_8	50
user_9	48
user_10	56
user_11	49
user_12	41
user_13	62
user_14	47
user_15	38
user_16	46
user_17	70
user_18	39
user_19	43
user_20	43
user_21	45

user_4	
user_1	41
user_2	44
user_3	46
user_4	88
user_5	39
user_6	53
user_8	45
user_9	57
user_10	57
user_11	47
user_12	58
user_13	50
user_14	48
user_15	55
user_16	59
user_17	48
user_18	35
user_19	47
user_20	43
user_21	40

user_5	
user_1	40
user_2	59
user_3	38
user_4	61
user_5	83
user_6	52
user_8	55
user_9	46
user_10	34
user_11	39
user_12	44
user_13	48
user_14	52
user_15	43
user_16	52
user_17	54
user_18	52
user_19	52
user_20	51
user_21	45

user_6	
user_1	34
user_2	59
user_3	53
user_4	57
user_5	50
user_6	66
user_8	48
user_9	57
user_10	53
user_11	53
user_12	44
user_13	49
user_14	40
user_15	40
user_16	44
user_17	54
user_18	54
user_19	36
user_20	59
user_21	50

user_8	
user_1	52
user_2	68
user_3	44
user_4	50
user_5	54
user_6	57
user_8	61
user_9	41
user_10	49
user_11	40
user_12	48
user_13	57
user_14	49
user_15	59
user_16	41
user_17	40
user_18	51
user_19	43
user_20	49
user_21	47

user_9	
user_1	59
user_2	58
user_3	44
user_4	55
user_5	53
user_6	53
user_8	40
user_9	69
user_10	40
user_11	40
user_12	52
user_13	44
user_14	46
user_15	45
user_16	48
user_17	53
user_18	42
user_19	58
user_20	59
user_21	42

user_10	
user_1	52
user_2	50
user_3	42
user_4	46
user_5	52
user_6	49
user_8	58
user_9	61
user_10	73
user_11	65
user_12	53
user_13	49
user_14	46
user_15	41
user_16	40
user_17	34
user_18	50
user_19	50
user_20	45
user_21	44

user_11	
user_1	54
user_2	52
user_3	46
user_4	43
user_5	52
user_6	50
user_8	45
user_9	47
user_10	47
user_11	57
user_12	40
user_13	54
user_14	43
user_15	41
user_16	49
user_17	48
user_18	45
user_19	105
user_20	45
user_21	37

user_12	
user_1	49
user_2	55
user_3	50
user_4	49
user_5	52
user_6	46
user_8	42
user_9	52
user_10	46
user_11	44
user_12	63
user_13	46
user_14	44
user_15	45
user_16	60
user_17	62
user_18	44
user_19	49
user_20	56
user_21	46

user_13	
user_1	50
user_2	61
user_3	41
user_4	57
user_5	52
user_6	50
user_8	51
user_9	37
user_10	53
user_11	46
user_12	50
user_13	81
user_14	46
user_15	47
user_16	51
user_17	42
user_18	44
user_19	47
user_20	50
user_21	44

user_14	
user_1	39
user_2	52
user_3	53
user_4	47
user_5	33
user_6	54
user_8	51
user_9	61
user_10	48
user_11	43
user_12	60
user_13	39
user_14	82
user_15	36
user_16	48
user_17	49
user_18	53
user_19	52
user_20	49
user_21	51

user_15	
user_1	45
user_2	39
user_3	45
user_4	53
user_5	55
user_6	46
user_8	58
user_9	61
user_10	52
user_11	47
user_12	57
user_13	44
user_14	51
user_15	78
user_16	45
user_17	33
user_18	41
user_19	40
user_20	64
user_21	46

user_16	
user_1	50
user_2	39
user_3	60
user_4	39
user_5	47
user_6	56
user_8	45
user_9	46
user_10	46
user_11	44
user_12	47
user_13	37
user_14	46
user_15	43
user_16	72
user_17	124
user_18	42
user_19	38
user_20	49
user_21	30

user_17	
user_1	38
user_2	51
user_3	46
user_4	52
user_5	42
user_6	45
user_8	46
user_9	41
user_10	57
user_11	49
user_12	38
user_13	46
user_14	54
user_15	42
user_16	105
user_17	43
user_18	44
user_19	44
user_20	61
user_21	56

user_18	
user_1	46
user_2	89
user_3	43
user_4	35
user_5	33
user_6	39
user_8	44
user_9	42
user_10	43
user_11	32
user_12	47
user_13	45
user_14	42
user_15	39
user_16	41
user_17	45
user_18	37
user_19	40
user_20	122
user_21	96

user_19	
user_1	46
user_2	43
user_3	48
user_4	50
user_5	44
user_6	49
user_8	51
user_9	55
user_10	49
user_11	100
user_12	48
user_13	41
user_14	51
user_15	45
user_16	59
user_17	36
user_18	46
user_19	48
user_20	43
user_21	48

user_20	
user_1	43
user_2	73
user_3	49
user_4	39
user_5	52
user_6	41
user_8	35
user_9	42
user_10	35
user_11	47
user_12	59
user_13	43
user_14	42
user_15	54
user_16	49
user_17	36
user_18	89
user_19	33
user_20	44
user_21	95

user_21	
user_1	38
user_2	106
user_3	37
user_4	45
user_5	38
user_6	46
user_8	50
user_9	35
user_10	44
user_11	43
user_12	44
user_13	41
user_14	47
user_15	34
user_16	34
user_17	49
user_18	84
user_19	51
user_20	102
user_21	32

Tabla 5.3 – Número de consultas enviadas al WSE y creadas por un determinado usuario

A primera vista, se puede comprobar como las consultas se distribuyen de forma casi equitativa entre todos los usuarios de la red.

Otro primer dato interesante que se puede extraer de los anteriores resultados, es que los usuarios con mayor número de amigos envían menos consultas directamente al WSE.

Además, fijándose en los amigos del usuario que crea la consulta, éstos envían más consultas directamente al WSE que aquellos que no tienen relación directa con el usuario. Y dentro del grupo de amigos, aquellos con menor número de vecinos son los que envían más consultas directamente al WSE. Esto es lógico, ya que aquellos amigos con menor número de vecinos tienen mayor probabilidad de enviar la consulta directamente al WSE en caso de que sus vecinos rechacen la consulta.

Todos estos datos son un reflejo de la teoría del protocolo, que asigna una mayor probabilidad de envío de una consulta a aquellos vecinos con un mayor número de amigos. En concreto están relacionados con la función de selección de usuario (*Sección 2.3.2*) y con la función de reenvío (*Sección 2.3.3*).

Por lo que teniendo en cuenta la teoría del protocolo y los resultados obtenidos, se puede concluir que el esquema diseñado funciona correctamente en caso de que todos los usuarios sigan el protocolo, y por lo tanto pueden mantener su privacidad con respecto al WSE, que lo único que conseguirá será obtener un perfil distorsionado de los miembros de la red social.

5.2.2 Comportamiento egoísta

Se realizará la misma prueba que la anterior, pero ahora simulando que el "*user_18*" rechace todas las consultas que le envíen sus vecinos.

Consultas enviadaspor usuarios sin relación directa

Consultas enviadas al WSE por amigos del usuario

Consultas propias enviadas directamente al WSE

user_1	
user_1	30
user_2	118
user_3	38
user_4	27
user_5	35
user_6	34
user_8	28
user_9	32
user_10	41
user_11	72
user_12	33
user_13	38
user_14	29
user_15	29
user_16	78
user_17	69
user_18	0
user_19	69
user_20	113
user_21	87

user_2	
user_1	52
user_2	41
user_3	44
user_4	51
user_5	53
user_6	56
user_8	50
user_9	40
user_10	41
user_11	42
user_12	50
user_13	59
user_14	35
user_15	50
user_16	50
user_17	53
user_18	0
user_19	44
user_20	85
user_21	104

user_3	
user_1	59
user_2	56
user_3	70
user_4	49
user_5	59
user_6	42
user_8	44
user_9	57
user_10	65
user_11	40
user_12	44
user_13	61
user_14	37
user_15	46
user_16	60
user_17	48
user_18	0
user_19	47
user_20	54
user_21	62

user_4	
user_1	42
user_2	55
user_3	39
user_4	80
user_5	56
user_6	55
user_8	53
user_9	62
user_10	44
user_11	41
user_12	63
user_13	46
user_14	47
user_15	36
user_16	56
user_17	58
user_18	0
user_19	65
user_20	58
user_21	44

user_5	
user_1	49
user_2	58
user_3	52
user_4	47
user_5	63
user_6	59
user_8	48
user_9	43
user_10	60
user_11	52
user_12	61
user_13	47
user_14	50
user_15	50
user_16	36
user_17	64
user_18	0
user_19	52
user_20	58
user_21	51

user_6	
user_1	56
user_2	55
user_3	54
user_4	50
user_5	64
user_6	78
user_8	48
user_9	55
user_10	48
user_11	45
user_12	47
user_13	55
user_14	53
user_15	47
user_16	46
user_17	53
user_18	0
user_19	58
user_20	54
user_21	34

user_8	
user_1	49
user_2	53
user_3	51
user_4	53
user_5	64
user_6	51
user_8	85
user_9	55
user_10	42
user_11	42
user_12	50
user_13	46
user_14	41
user_15	61
user_16	55
user_17	52
user_18	0
user_19	44
user_20	49
user_21	57

user_9	
user_1	68
user_2	58
user_3	60
user_4	63
user_5	43
user_6	41
user_8	46
user_9	62
user_10	58
user_11	60
user_12	48
user_13	52
user_14	58
user_15	39
user_16	49
user_17	49
user_18	0
user_19	50
user_20	53
user_21	43

user_10	
user_1	57
user_2	53
user_3	50
user_4	54
user_5	51
user_6	41
user_8	60
user_9	60
user_10	69
user_11	45
user_12	56
user_13	47
user_14	49
user_15	51
user_16	57
user_17	51
user_18	0
user_19	55
user_20	50
user_21	44

user_11	
user_1	54
user_2	55
user_3	57
user_4	37
user_5	58
user_6	50
user_8	47
user_9	54
user_10	53
user_11	52
user_12	43
user_13	42
user_14	45
user_15	51
user_16	50
user_17	43
user_18	0
user_19	104
user_20	54
user_21	51

user_12	
user_1	56
user_2	47
user_3	45
user_4	53
user_5	58
user_6	55
user_8	44
user_9	56
user_10	60
user_11	58
user_12	78
user_13	49
user_14	50
user_15	42
user_16	46
user_17	54
user_18	0
user_19	57
user_20	57
user_21	35

user_13	
user_1	62
user_2	57
user_3	53
user_4	48
user_5	56
user_6	49
user_8	57
user_9	48
user_10	35
user_11	53
user_12	59
user_13	73
user_14	57
user_15	42
user_16	33
user_17	52
user_18	0
user_19	55
user_20	63
user_21	48

user_14	
user_1	47
user_2	65
user_3	51
user_4	45
user_5	41
user_6	54
user_8	59
user_9	40
user_10	53
user_11	46
user_12	57
user_13	54
user_14	78
user_15	61
user_16	49
user_17	48
user_18	0
user_19	47
user_20	59
user_21	46

user_15	
user_1	57
user_2	49
user_3	49
user_4	54
user_5	67
user_6	51
user_8	55
user_9	51
user_10	37
user_11	49
user_12	42
user_13	49
user_14	45
user_15	78
user_16	49
user_17	52
user_18	0
user_19	60
user_20	51
user_21	55

user_16	
user_1	53
user_2	44
user_3	54
user_4	43
user_5	47
user_6	43
user_8	54
user_9	44
user_10	56
user_11	59
user_12	53
user_13	47
user_14	48
user_15	51
user_16	60
user_17	103
user_18	0
user_19	49
user_20	53
user_21	39

user_17	
user_1	49
user_2	48
user_3	55
user_4	54
user_5	54
user_6	58
user_8	41
user_9	53
user_10	48
user_11	52
user_12	46
user_13	54
user_14	46
user_15	43
user_16	93
user_17	59
user_18	0
user_19	39
user_20	51
user_21	57

user_18	
user_1	1
user_2	4
user_3	1
user_4	1
user_5	0
user_6	0
user_8	1
user_9	2
user_10	0
user_11	1
user_12	1
user_13	0
user_14	0
user_15	1
user_16	0
user_17	1
user_18	976
user_19	2
user_20	6
user_21	2

user_19	
user_1	48
user_2	42
user_3	59
user_4	42
user_5	51
user_6	42
user_8	46
user_9	53
user_10	57
user_11	117
user_12	64
user_13	52
user_14	45
user_15	58
user_16	55
user_17	47
user_18	0
user_19	39
user_20	48
user_21	35

user_20	
user_1	40
user_2	77
user_3	50
user_4	42
user_5	56
user_6	43
user_8	57
user_9	50
user_10	50
user_11	50
user_12	42
user_13	52
user_14	47
user_15	55
user_16	43
user_17	44
user_18	0
user_19	41
user_20	57
user_21	104

user_21	
user_1	53
user_2	107
user_3	48
user_4	33
user_5	45
user_6	49
user_8	49
user_9	42
user_10	45
user_11	53
user_12	44
user_13	54
user_14	53
user_15	41
user_16	58
user_17	51
user_18	0
user_19	43
user_20	87
user_21	45

Tabla 5.4 – Número de consultas enviadas al WSE por cada usuario y creadas por un usuario en concreto

Se puede observar claramente que el usuario "*user_18*" rechaza todas las consultas de sus vecinos, por lo que todas las tablas reflejan que el "*user_18*" no envía al WSE ninguna consulta ajena.

Además, observando la tabla de este usuario, se puede ver cómo otros usuarios de la red envían alguna de las consultas creadas por "*user_18*" al WSE. Estas consultas son de las primeras creadas por el usuario, ya que posteriormente los demás usuarios al ver que el "*user_18*" rechaza todas sus consultas, van disminuyendo la probabilidad de aceptar consultas que procedan de él, y de esta forma obligan a "*user_18*" a enviar directamente sus consultas al WSE.

Como se puede ver en la tabla del usuario, de 1000 consultas creadas, envía 976 él mismo al WSE. Esto indica que el protocolo lo ha aislado por su comportamiento egoísta, hecho que hace que su privacidad se vea expuesta, permitiendo al WSE crear un perfil del usuario.

De toda la información extraída de los resultados, se concluye que el esquema implementado también funciona adecuadamente cuando algún usuario se comporta de forma egoísta y no sigue el protocolo.

5.3 Número de saltos

En este test se analizarán el número de saltos en redes de diferente tamaño. Es interesante saber si el número de saltos se mantiene o en cambio aumenta o disminuye si el protocolo está funcionando en una red social con mayor número de usuarios.

Para ello, se utilizará la red social de 20 usuarios y la red de 100 usuarios. Esta última red también ha sido elaborada a partir de la red Twitter de 20 usuarios, creando manualmente nuevos usuarios con relaciones de amistad aleatorias. El grafo de esta red se encuentra descrito en un fichero ".dl" de igual forma que la red social de 20 usuarios y que se necesitará para cargar el grafo en la base de datos.

5.3.1 Red de 20 usuarios

Calculando la media de los resultados obtenidos después de 5 pruebas, en las que en cada una cada usuario envía 1000 consultas, se obtienen los siguientes datos:

- Número medio de saltos por consulta: 2,11345

Teniendo en cuenta este resultado, se puede deducir que las consultas no viajan demasiado por la red, por lo que se podría estimar que el tiempo de comunicación entre usuarios tampoco será demasiado elevado.

5.3.2 Red de 100 usuarios

Cada usuario envía 100 consultas en cada una de las 5 pruebas realizadas. La media obtenida con respecto al número de saltos es la siguiente:

- Número medio de saltos por consulta: 3,09345

Se puede observar que aumentando el número de usuarios de la red, también ha aumentado el número medio de saltos de una consulta, lo que conlleva además un mayor tiempo total de búsqueda.

5.4 Tiempo de proceso del usuario

Es importante evaluar también el tiempo que tardaría un usuario en enviar una consulta y recibir la correspondiente respuesta en una situación real. Para ello, se han realizado varios test sobre la red de 20 usuarios. En cada una de ellas cada usuario envía 1000 consultas y todos siguen un comportamiento normal, es decir, respetando el protocolo.

Realizando una media de los resultados obtenidos en todas las pruebas se han obtenido estos resultados:

- Tiempo medio por consulta: 0.26088 segundos
- Número medio de saltos por consulta: 2,11525
- Tiempo medio de proceso de cada usuario: 0,105488 segundos

La estimación del tiempo de una consulta en un entorno real se calcularía de la siguiente forma:

$$t_{consulta} = t_{procesoUsuario} * (n^o\ medio\ saltos + 1) + 2 * (t_{salto}) * (n^o\ medio\ saltos) + t_{respuestaWSE}$$

El tiempo de proceso de usuario es el que va desde que un determinado usuario recibe (o crea) una consulta hasta que la envía a uno de sus vecinos (o al buscador). El número medio de saltos de una consulta va ligado con el número medio de usuarios que la procesan, ya que para calcular este dato sólo es necesario sumar una unidad al número medio de saltos. Es decir, si una consulta realiza dos saltos, entonces la consulta es procesada por tres usuarios, tal y como muestra la *Figura 5.1*.

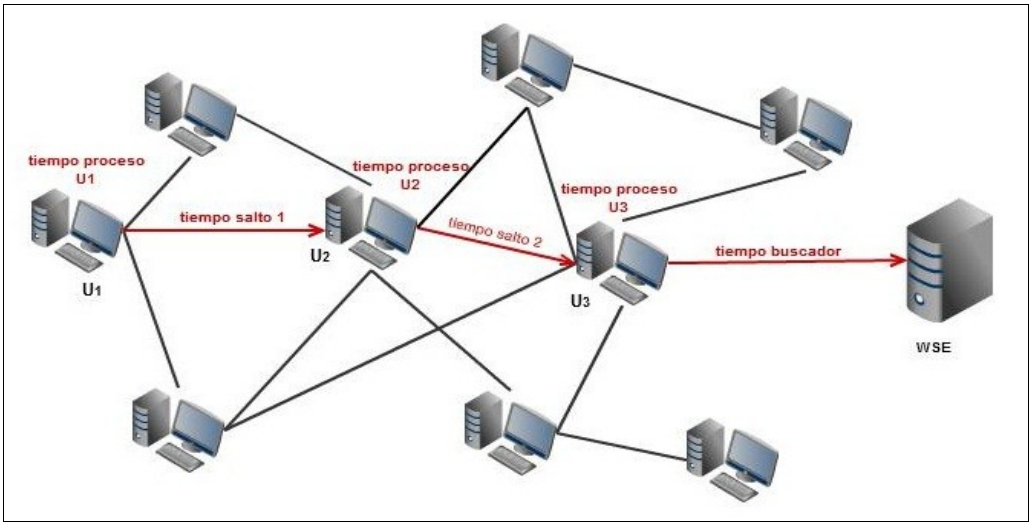


Figura 5.1 – Variables que influyen en el cálculo del tiempo de una consulta en un entorno real

Para la estimación es necesario tener en cuenta que el tiempo de cada salto debe ser multiplicado por dos (tal y como se puede observar en la fórmula anterior), ya que también se debe contabilizar el tiempo de retorno de la consulta.

El estudio presentado en [13] establece que el tiempo medio de comunicación entre dos usuarios en una red P2P es de 530 ms. En [14] se estima que el tiempo medio que necesita un usuario para enviar una consulta al WSE y recibir su correspondiente respuesta es de 400 ms. Con estos datos y los obtenidos en la

prueba, se puede realizar la estimación utilizando la fórmula anterior (resultado en segundos):

$$t_{consulta} = 0.105488 * (2.11525 + 1) + 2 * (0.530) * (2.11525) + 0.4 = 2.9708$$

La propuesta similar más rápida es *UUP* [11], que necesita 5,2 segundos para realizar una búsqueda utilizando un grupo de tres usuarios y una clave de 1024 bits. Otros tipos de esquemas que se basan en *Tor* [12] necesitan de media 10 segundos.

Atendiendo a estos resultados, la estimación obtenida se puede considerar como muy buena si se compara con otras implementaciones.

Capítulo 6

Conclusiones

En este capítulo se analiza el concepto de privacidad bajo el punto de vista de este proyecto (apartado 6.1), y se extraen conclusiones analizando los resultados de la implementación (apartado 6.2). En la sección 6.3 se describen los problemas encontrados durante la elaboración del proyecto y para terminar, el apartado 6.4 presenta algunas ideas para un posible trabajo futuro.

6.1 Privacidad

La privacidad es un aspecto muy a tener en cuenta en la Internet actual. Los blogs, las redes sociales, los buscadores de Internet, son herramientas útiles para los usuarios pero a la vez peligrosas si no se hace un uso adecuado de ellas. No hay más que navegar un poco por cualquier red social para darse cuenta de que muchos usuarios no tienen en cuenta su nivel de privacidad, y los riesgos que eso conlleva.

Pero el tema que aborda el actual proyecto va más allá. En la mayoría de las páginas de inicio de los usuarios, aparece un buscador de Internet, ya que su uso es imprescindible para encontrar aquello que se necesita en la red. Pero detrás de esta funcionalidad, se esconde una máquina capaz de archivar e indexar todo aquello que busque un determinado usuario, elaborando un perfil y capaz de obtener la información más íntima de un usuario.

Se podía prever que para ofrecer una mejor calidad de servicio, los WSE guardaban cierta información de los usuarios, pero es difícil poder conocer los entresijos de los algoritmos de los WSE y realmente determinar su verdadero funcionamiento.

No fue hasta después del escándalo de AOL [9], que se pudo comprobar qué información obtenían y cómo elaboraban un perfil de cada usuario. Este hecho escandalizó a los usuarios preocupados por su privacidad, ya que echando un vistazo a los "*logs*" de AOL, se pueden conocer datos tan privados de un usuario como su nombre y apellidos, DNI, número de la seguridad social, aficiones, preferencias sexuales, estado civil...

Quizás haya muchos usuarios que puedan pensar que no les preocupa que el WSE tenga sus datos personales, pero habría que pensar qué puede hacer el WSE con esos datos. Está claro que el perfil del usuario sirve para ofrecer un mejor servicio y más personalizado, pero estos datos del usuario tienen un valor económico. Y ahí es el WSE el que puede comercializar con dicha información, negociando con cualquier empresa que necesite datos de usuarios para cualquier propósito. Puede ser un caso extremo, pero se debe poner en la peor situación, porque realmente nadie sabe qué ocurre detrás de las páginas de los buscadores.

Se puede pensar en la existencia de la ley de protección de datos y otras leyes que obligan a terceros a mantener los datos de los usuarios bajo protección, impidiendo que se saque provecho de los datos personales y venderlos a otras empresas. Sin embargo, aunque esté prohibido por ley, nadie puede asegurar que no se vaya a hacer, por lo que un primer paso para evitar estos casos es el de proteger la privacidad de un usuario en la medida de lo posible.

Y ya no sólo lo que haga el WSE con el perfil del usuario, sino que también puede ocurrir cualquier error o problema en los servidores de los WSE que causen pérdidas de información tal y como pasó con AOL, y provocar que dicha información pueda estar accesible al público, pudiendo observar los datos de los perfiles de usuarios, con el riesgo que puede conllevar a algún usuario que haya realizado alguna búsqueda comprometida.

6.2 Resultados de la implementación

El proyecto ha servido para evaluar el comportamiento del protocolo implementado y comprobar que puede ser factible en un entorno real. Se han obtenido datos satisfactorios en cuanto a tiempo de búsqueda de las consultas, así como también en el número promedio de saltos que realizan las búsquedas.

En comparación con otras implementaciones similares que tratan de mantener la privacidad de los usuarios de los WSE, tales como Tor [12] o UUP [11], se han obtenido mejores resultados en cuanto a tiempo de búsqueda.

Quizás un aspecto que se debe estudiar en la implementación es el de la criptografía de clave pública. Los tiempos de proceso de cada usuario aumentan debido a la firma y verificación de las consultas, y aunque se obtienen buenos resultados en cuanto a tiempo con respecto a otras propuestas, si se prescindiese de la firma digital de las consultas el tiempo de proceso mejoraría.

Otro punto a destacar es que aunque en la implementación del protocolo no se ha incluido la verificación de la clave pública de un usuario, la existencia de una Autoridad Certificadora complicaría la puesta en marcha del protocolo en un escenario real, por el simple hecho de que cada usuario de la red debería poseer una clave pública certificada por una CA.

Indicar que la versión implementada del cliente es totalmente operativa en un entorno real, aunque para que funcione correctamente es necesario configurar las tablas NAT de los routers de los distintos nodos y disponer de una máquina que simule el comportamiento del servidor de una red social.

A nivel personal, este proyecto me ha servido para aprender cómo funcionan los buscadores de Internet y ligado a ello, de la importancia de la privacidad de los usuarios. En cuanto a la implementación del protocolo me ha ayudado a mejorar mis conocimientos de Java, además de darme cuenta de la importancia de seguir

unos pasos previos a la programación, como han sido el análisis y el diseño del protocolo.

6.3 Problemas encontrados

Durante la fase de programación del protocolo han ido apareciendo pequeños contratiempos:

- La aplicación lanzaba errores cuando existía una consulta que excedía del tiempo máximo de búsqueda establecido. En estos casos el usuario que creaba la consulta cerraba el *socket* con el usuario al que le había enviado la consulta y seleccionaba otro en su lugar para reenviar la consulta. El error fue solucionado y era debido a que los nodos intermedios no cerraban sus canales de comunicación adecuadamente.
- Se encontraron problemas a la hora de enviar objetos a través de los *sockets*. Se solucionaron haciendo que las clases que se querían enviar a través de la red implementaran la interfaz "*Serializable*" de Java.
- Ejecutando el protocolo, se observó que una consulta reenviada por un usuario podía recibirla de nuevo a través de uno de sus vecinos para volver a procesarla. Este hecho aumentaba el número de saltos y consecuentemente el tiempo de búsqueda. Finalmente se optó por evitar esta situación, impidiendo que un usuario participase más de una vez en el proceso de una misma consulta. Para ello, se dotó a cada consulta de un identificador único de 128 bits, generado de forma aleatoria, y haciendo que cada usuario que tratase una consulta añadiese su identificador a una tabla en memoria de consultas en proceso. De esta forma, siempre que un usuario recibiese una consulta, podía verificar si ya la había tratado o no. En estos casos se rechazaba la consulta indicando el motivo, y no se actualizaban las probabilidades relativas al nivel de egoísmo.
- Realizando pruebas con el protocolo, había ocasiones en los que algunos WSE rechazaban las peticiones debido al envío masivo de consultas que se estaba llevando a cabo.

6.4 Trabajo futuro

Un posible trabajo futuro fuera del alcance de este proyecto, sería el de poder integrar el protocolo en una red social existente, como Twitter o Facebook y poder beneficiarse de sus protocolos de comunicación y de los grupos de usuarios creados.

Otro aspecto que se podría tener en cuenta, es el de intentar mejorar aún más el tiempo de búsqueda de una consulta. Una primera opción sería el de mejorar el tiempo de proceso de cada nodo. En este sentido se podría suprimir la firma digital y su verificación, tal y como se ha comentado anteriormente, ya que el tiempo de proceso mejoraría, aunque habría que destacar que en caso de que alguna consulta vulnerara la ley, no se podría identificar quién es el creador de la consulta.

Otra opción para mejorar el tiempo de proceso de usuario podría ser la de establecer un número máximo de saltos por consulta. En las pruebas realizadas, se podía comprobar que una consulta podía realizar bastantes saltos si los usuarios que la procesaban tenían muchos amigos. Estableciendo un número de saltos máximo se reduciría el tiempo en estas búsquedas, aún así, se debería estudiar si esta opción mantiene la distribución equitativa de consultas y no expone la privacidad de ningún usuario.

Sería también interesante conseguir un grafo de una red social real de mayor tamaño que las utilizadas en este proyecto, por ejemplo de 500 o 1000 usuarios, y evaluar cómo aumenta el número de saltos de las consultas (crecimiento lineal, exponencial, etc..).

7. Bibliografía

[1] Iprospect.com, Inc., Iprospect Blended Search Results Study <http://www.iprospect.com> (2008).

[2] “Exploiting Social Networks to Provide Privacy in Personalized Web Search”, A. Erola, J. Castellà-Roca, A. Viejo, J.M. Mateo-Sanz (2011)

[3] “Using social networks to distort users' profiles generated by web search engines”, Alexandre Viejo, Jordi Castellà-Roca (2009)

[4] European Commission, The legal and Market Aspects of Electronic Signatures <http://ec.europa.eu> (2009)

[5] European Parliament and Council, Directive on the Retention of Data Generated or Processed in Connection with the Provision of Publicly Available Electronic Communications Services or of Public Communications Networks, <http://eur-lex.europa.eu/> (2006)

[6] http://jxse.kenai.com/Tutorials/JXSE_ProgGuide_v2.5.pdf

[7] http://www.w3.org/TR/xmldsig-core/

[8] http://java.sun.com/developer/technicalArticles/xml/dig_signature_api/

[9] http://en.wikipedia.org/wiki/AOL_search_data_scandal (2006)

[10] Aol, AOL Keyword Searches, http://www.gregsadetsky.com/aol-data/

[11] J. Castellà-Roca, A. Viejo, J. Herrera-Joancomartí, Preserving user's privacy in web search engines, Computer Communications 32 (13-14) (2009) 1541-1551.

[12] The Tor Project, http://www.torproject.org>, 2009

[13] D. Choffnes, F. Bustamante, "Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems, in: SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication, 2008, pp. 363-374.

[14] J. Castellà-Roca, A. Viejo, J. Herrera-Joancomartí, Preserving user's privacy in web search engines, Computer Communications 54 (2010) 1343-1357.

[15] J.Domingo-Ferrer, M. Bras-Amorós, Q. Wu, J.Manjón, User-private information retrieval based on a peer-to-peer community, Data and Knowledge Engineering 68 (2009) 1237-1252

[16] F. Saint-Jean, A. Johnson, D. Boneh, J. Feigenbaum, Private web search, in: Proceedings of the ACM Workshop on Privacy in Electronic Society – WPES'07, 2007, pp 84-90.

8. Anexo 1

Instalación de la aplicación

En este anexo se detallarán los pasos previos necesarios para el funcionamiento del protocolo en una máquina local.

8.1 Base de datos

Para comenzar, es necesario instalar un servidor de BBDD. En este caso se ha optado por descargar la aplicación gratuita XAAMP, que entre otras incluye el servidor web apache y el servidor de BBDD MySQL. El servidor web se ha empleado para poder utilizar la herramienta PHPMyAdmin que permite gestionar y configurar MySQL a través de una interfaz web como se verá más adelante.

XAAMP se puede descargar desde http://www.apachefriends.org/es/xampp.html.

Una vez instalado, se deben iniciar los servicios Apache y MySQL pulsando en el botón "*Start*" correspondiente tal y como muestra la *Figura 8.1*.

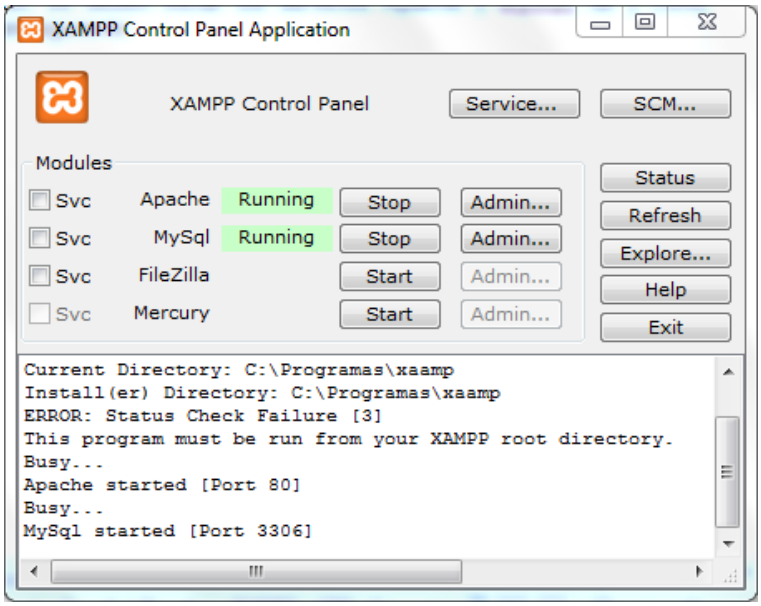


Figura 8.1 – Inicio de los servicios Apache y MySql

Ahora se procederá a configurar la base de datos. Recordar que esta base de datos simulará la BBDD del servidor de la red social. Para ello, se utilizará la herramienta *PHPMyAdmin*, por lo que se debe abrir en una ventana del explorador de Internet la siguiente dirección: http://localhost/phpmyadmin/index.php. Entonces aparecerá una ventana como la que se muestra en la *Figura 8.2*.

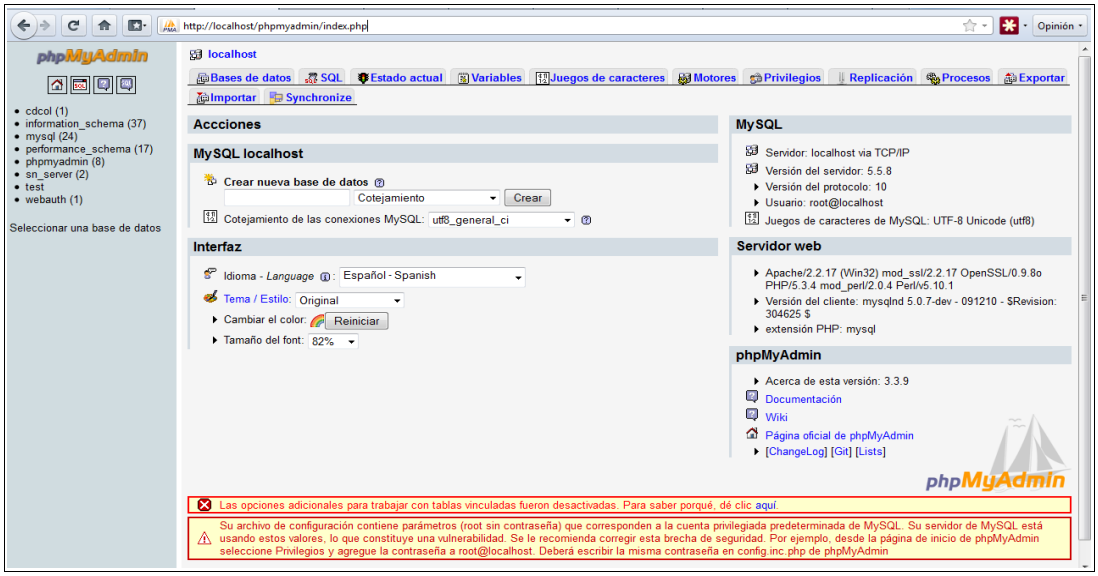


Figura 8.2 – Interfaz web de PHPMyAdmin

En la página principal ya se ofrece la posibilidad de crear un nueva base de datos. Se debe introducir el nombre "*sn_server*" y pulsar "*Crear*".

En este punto, ya se tiene creada la base de datos, pero es necesario definir un usuario con privilegios para que pueda interactuar con ella. Para ello, en la parte superior de la interfaz aparecerá una pestaña denominada "*Privilegios*" que se deberá seleccionar. En la nueva página aparecerá una opción que indica "*Agregar un nuevo usuario*", resaltada en la *Figura 8.3*.

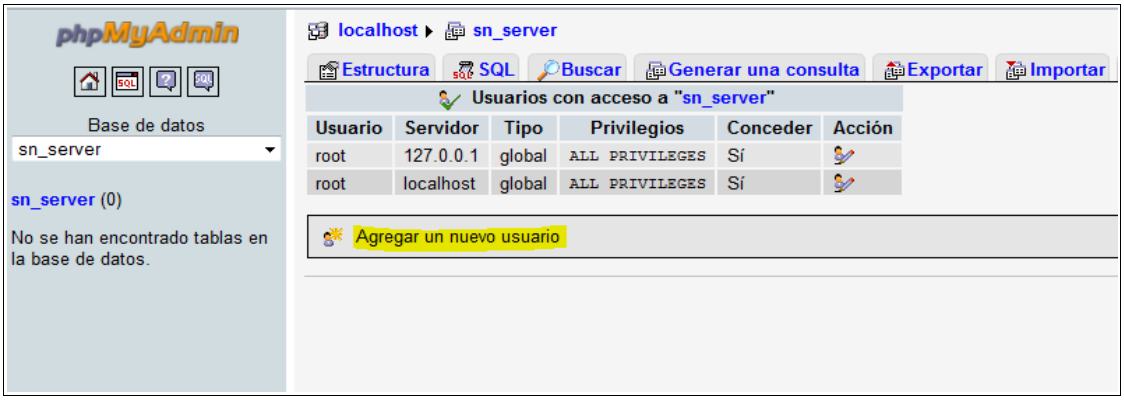


Figura 8.3 – Apartado de "Privilegios" de la base de datos "sn_server"

Haciendo *click* sobre el enlace se abrirá una nueva página que permitirá crear el nuevo usuario y configurar sus privilegios. Se debe introducir el nombre de usuario "*user*", el nombre del servidor "*localhost*" y como contraseña "*user*", tal y como se puede observar en la *Figura 8.4*.

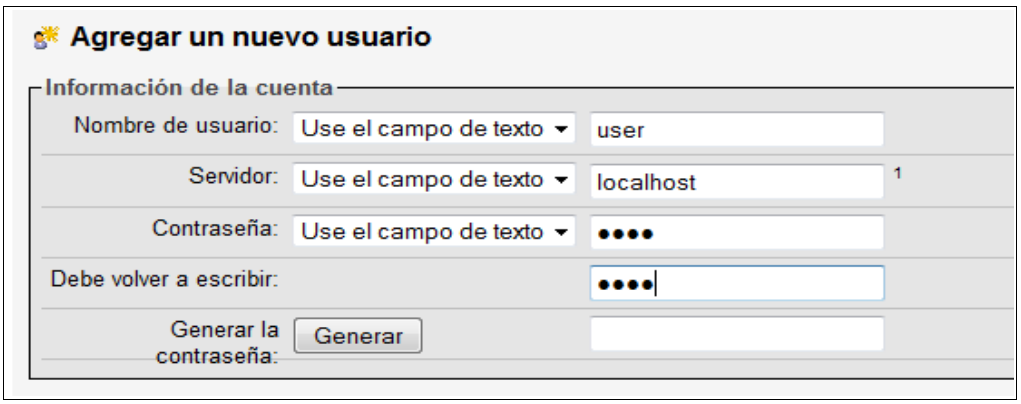


Figura 8.4 – Formulario para agregar un nuevo usuario a la base de datos

En la misma página se marca la opción "*Grant all privileges on database sn_server*" y se deja sin marcar los privilegios globales, tal y como muestra la *Figura 8.5*.

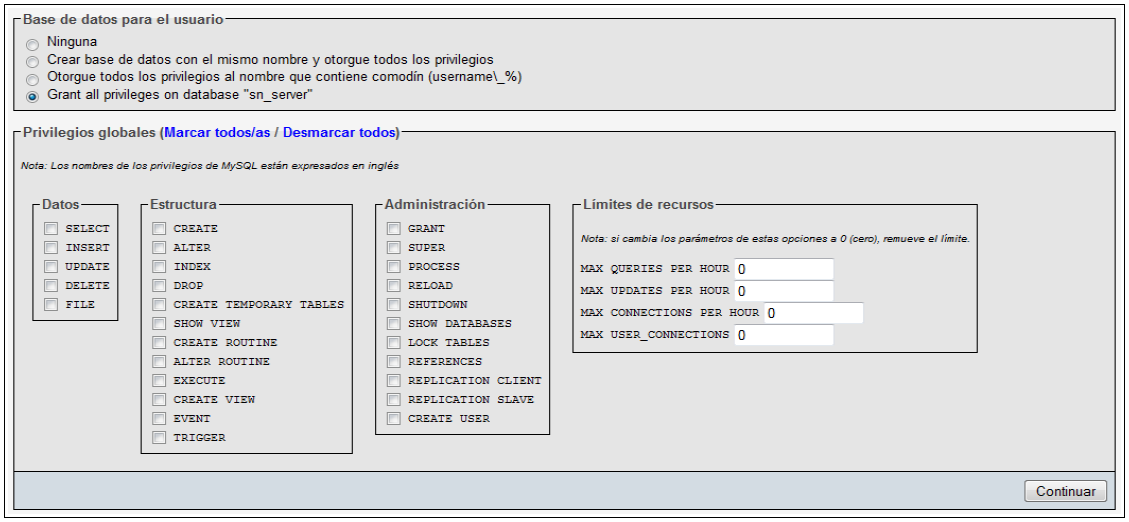


Figura 8.5 – Selección de los privilegios del nuevo usuario

Tras pulsar el botón "*Continuar*" ya se habrá creado el nuevo usuario con los privilegios adecuados sobre la BBDD "*sn_server*".

Ahora es necesario crear las tablas de la base de datos. Para realizar este paso se debe importar un fichero que contiene las sentencias SQL apropiadas. Este fichero irá adjunto al código fuente de la aplicación y se denomina "*script_BBDD.sql*". El contenido de este archivo es el mostrado en la *Figura 8.6*.

```
1   DROP TABLE IF EXISTS FRIEND;
2   DROP TABLE IF EXISTS USER;
3
4   CREATE TABLE USER (
5     USER_ID INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
6     NICK VARCHAR(25) NULL,
7     IP VARCHAR(15) NULL,
8     PORT INTEGER UNSIGNED NULL,
9     STATUS INTEGER UNSIGNED NULL,
10    N_FRIENDS INTEGER UNSIGNED NULL,
11    PRIMARY KEY(USER_ID)
12  );
13
14
15  CREATE TABLE FRIEND (
16    USER1_ID INTEGER UNSIGNED NOT NULL,
17    USER2_ID INTEGER UNSIGNED NOT NULL,
18    PRIMARY KEY (USER1_ID, USER2_ID),
19    FOREIGN KEY (USER1_ID)
20      REFERENCES USER(USER_ID)
21        ON DELETE NO ACTION
22        ON UPDATE NO ACTION,
23    FOREIGN KEY (USER2_ID)
24      REFERENCES USER(USER_ID)
25        ON DELETE NO ACTION
26        ON UPDATE NO ACTION
27  );
```

Figura 8.6 – Contenido del fichero "script_BBDD.sql"

Para realizar la importación del script, se debe seleccionar la base de datos que se ha creado, denominada "*sn_server*", en el menú de la izquierda, y después pulsar sobre la pestaña "Importar". A través del botón "*Examinar*" se deberá buscar y seleccionar el archivo "*script_BBDD.sql*" en el directorio en el que esté almacenado y luego pulsar sobre el botón continuar.

Si todo ha salido correctamente aparecerá un mensaje informando del estado de la importación y se podrá observar en el menú de la izquierda como aparecen las nuevas tablas, "*friend*" y "*user*", como se puede ver en la *Figura 8.7*.

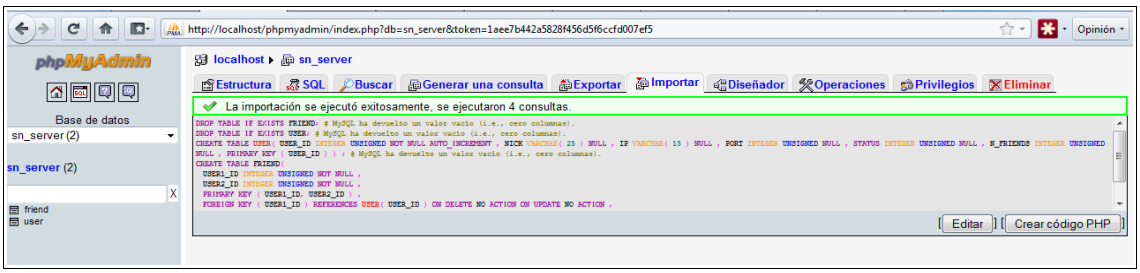


Figura 8.7– Confirmación del estado de la importación

8.2 Importación de la red social

Una vez creada la base de datos y sus tablas, se puede proceder a cargar los datos de los usuarios y sus relaciones de amistad.

Durante las pruebas se han utilizado dos redes sociales, descritas en los ficheros “Red_20usuarios.dl” y “Red_100usuarios.dl” que se pueden encontrar en el directorio del código fuente que acompaña a esta memoria.

Estos ficheros contienen las relaciones de los usuarios dentro de la red social, y para poder pasar esos datos a la base de datos es necesario utilizar la aplicación “ReadGraph”. Para ello es necesario situarse en el directorio “readGraph/bin” en donde se encuentra el código compilado de la aplicación y ejecutar:

```
# java -classpath ./mysql-connector-java-5.1.15-bin.jar; ReadGraphGUI
```

Como se puede observar, en el “*classpath*” se incluye el driver JDBC (Java Database Connectivity) de MySQL que permite conectar la aplicación Java con la base de datos. Tras ejecutar el comando anterior aparece la interfaz de la aplicación que es la mostrada en la *Figura 8.8*.

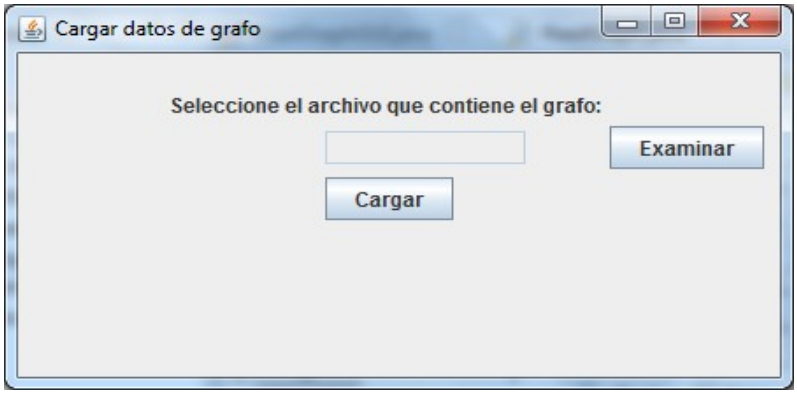


Figura 8.8 – Interfaz de la aplicación ReadGraph

Pulsando el botón “Examinar” se podrá seleccionar el fichero que contiene el grafo que se desea importar a la base de datos. En este caso se importará el archivo “*Red_20usuarios.dl*”. Esta situación se observa en la *Figura 8.9*.

Figura 8.9 – Selección del fichero que contiene los datos a importar

Una vez que se pulse el botón “*Cargar*” comenzará la lectura de los datos del fichero y la introducción de la información correspondiente en la base de datos. Al finalizar la importación aparecerá el mensaje que se puede ver en la *Figura 8.10*.

Figura 8.10 – Finalización exitosa de la importación

9. Anexo 2

Ejecución de un test

En esta sección se describirá cómo poner en marcha la aplicación para poder realizar pruebas con el protocolo en una máquina local.

9.1 Inicio del servidor de la red social

Antes de realizar pruebas con el protocolo es necesario iniciar el servidor que simulará el servidor de la red social. Para realizar este paso se debe abrir una ventana del "*Procesador de Comandos de Windows*" (cmd). Después hay que situarse en el directorio "PFC/bin/" que es en donde se encuentra el código compilado de la aplicación y ejecutar la clase "*Server*" utilizando el siguiente comando:

```
# java -classpath ./mysql-connector-java-5.1.15-bin.jar; Server
```

Por defecto, la aplicación está configurada para iniciar el servidor en el puerto 12300. Si se deseara modificar este valor sería necesario editar el valor de la variable "*serverPort*" de la clase "*Server.java*" y volver a compilar el fichero.

Una vez ejecutado el comando aparecerá un mensaje en pantalla indicando si el servidor ha sido iniciado o no. Este paso se puede observar en la *Figura 9.1*.

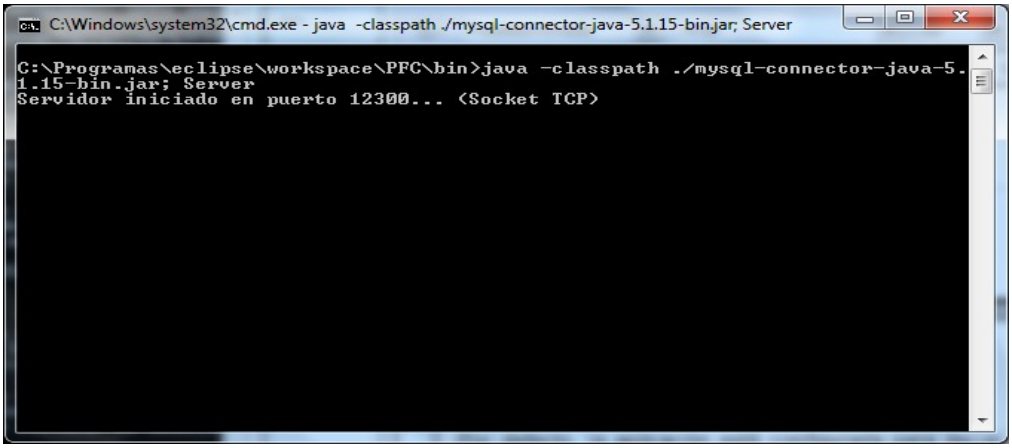


Figura 9.1 – Inicio del servidor

Para terminar el servidor basta con pulsar la combinación de teclas “CTRL+C” o cerrar la ventana anterior.

9.2 Inicio del script de pruebas

Una vez se tiene la BBDD con los datos de la red social y el servidor iniciado, se puede proceder a iniciar el *script* para ejecutar las pruebas. Para ello es necesario abrir una nueva ventana del “*Procesador de comandos de Windows*” y ejecutar el siguiente comando:

```
# java -classpath ./mysql-connector-java-5.1.15-bin.jar; Test
```

El comando anterior iniciará iniciará la aplicación que permitirá realizar las pruebas. Los resultados en esta ocasión serán mostrados por pantalla. Si se deseara que todos los resultados fuesen guardados en un fichero en vez de ser mostrados por pantalla se debería ejecutar el mismo comando anterior pero añadiendo el nombre del archivo en el que se guardarán los resultados. Por ejemplo, se iniciará el *script* de pruebas guardando todos los resultados en el fichero “*salida.txt*”:

```
# java -classpath ./mysql-connector-java-5.1.15-bin.jar; Test salida.txt
```

9.3 Realización de una prueba

Lo que hace inicialmente el *script* es poner online todos los usuarios de la red social y cargar en memoria todas las consultas contenidas en el fichero "busquedasAOL.txt" situado en el mismo directorio "/PFC/bin".

Posteriormente aparece un pequeño menú ofreciendo la posibilidad de iniciar un test o en cambio salir de la aplicación. Este menú es el que se muestra en la *Figura 9.2*.

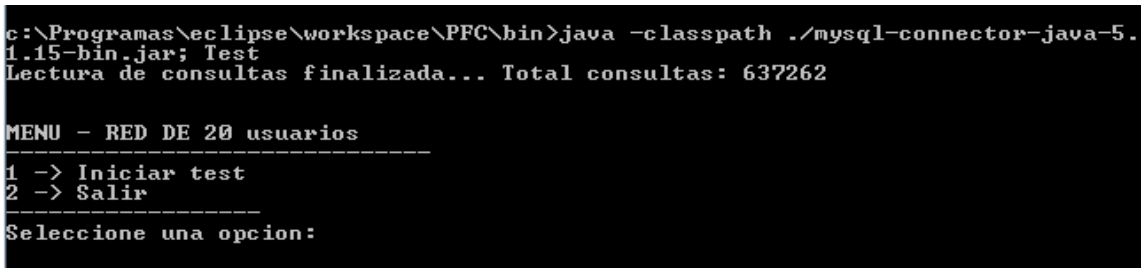


Figura 9.2 – Menú de la aplicación de Test

Pulsando "1" comenzará la ejecución de un test. Primeramente el *script* preguntará por el número de consultas que enviará cada usuario. Tras introducir este dato preguntará por el tiempo de pausa entre cada consulta. Esta pausa es necesaria para que no se solapen las consultas y que el procesador no deba ocuparse de más de una consulta a la vez, porque este hecho distorsionaría los resultados.

Una pausa de 400 milisegundos es suficiente para la red de 20 usuarios, mientras que para la red de 100 usuarios se debe introducir como mínimo una pausa de 4000 milisegundos, ya que al aumentar el número de saltos también aumenta el tiempo de proceso de cada consulta.

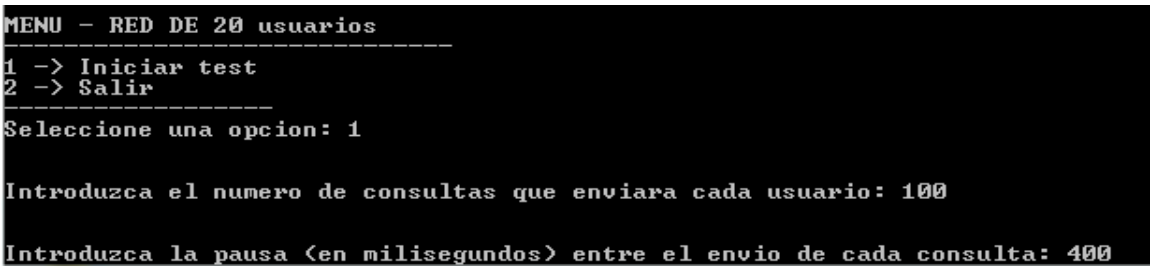


Figura 9.3 – Configuración de las opción de ejecución del test

Después de introducir estos datos comenzará el test. Por pantalla se podrá ver el proceso y al finalizar se mostrarán los resultados o se almacenarán en un fichero dependiendo de cómo se haya ejecutado el *script*.